Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN INGENIERÍA DE SOFTWARE

Aplicación Web para la Gestión de Adopción y Localización de Mascotas Perdidas

Estudiante: Christian Romarís Caamaño
Dirección: Manuel Álvarez Díaz

A Coruña, noviembre de 2019.

Dedicado a todos los han estado ahí y a los que non han podido estar.

Agradecimientos

A mi familia por darme la oportunidad de llegar hasta aquí. Especialmente a los que me han dado cobijo durante estos cuatro años.

A los amigos que me llevo y que me han hecho pasar unos de los mejores años de mi vida.

A Manuel Álvarez Díaz por ayudarme a completar este proyecto.

A los que que no han podido estar y sé que les habría gustado.

Resumen

Con este proyecto se pretende ayudar a mejorar la situación de muchos animales que se encuentran en estado de abandono o en diferentes protectoras. Para ello se ha desarrollado una aplicación Web que permita facilitar el proceso de adopción y la localización de animales perdidos.

La aplicación permite el registro de asociaciones de forma que puedan añadir animales que tengan en proceso de adopción. Los usuarios que accedan podrán localizar asociaciones cercanas y ver la información de animales en adopción. Además permite a los usuarios añadir animales perdidos indicando la localización en la que fueron vistos por última vez. De esta forma otros usuarios cercanos serán notificados y cualquier persona que acceda a la aplicación podrá indicar la posición de estos animales perdidos en caso de que sean vistos. Los dueños de animales perdidos tienen acceso a un mapa con las localizaciones en las que ha sido visto al animal.

La aplicación pretende reducir el esfuerzo que supone localizar animales en adopción que satisfagan los gustos de los usuarios. Por este motivo un usuario puede indicar las características que busca de un animal de forma que la aplicación lo notifique cuando se añada un animal que le pueda interesar. Además pueden recibir información por correo electrónico de estos animales. La plataforma también proporciona un chat para facilitar la comunicación entre usuarios.

Abstract

Palabras clave:

- Animal
- Adopción
- Localización
- React
- Aplicación Web
- Springboot
- Protectora

Keywords:

- Animal
- Adoption
- Location
- React
- Web Application
- Springboot
- Shelter

Índice general

Índice de figuras

Capítulo 1

Introdución

El aumento de la capacidad económica de la familias en los diferentes estratos sociales así como la necesidad de llenar espacios afectivos han provocado que, hoy en día, haya aumentado el número de mascotas en los entornos familiares.

La existencia de animales como miembros del núcleo familiar, se trata, por tanto, de un tema de actualidad. Resulta del todo normal encontrarse en cualquier tipo de vivienda, en cualquier parte del mundo, con mascotas, consideradas como un miembro más de las familias, que aportan una compañía insustituible. Como resultado del aumento de esta tendencia, el número de animales en las calles se ha visto incrementado por diferentes motivos: en ocasiones son abandonados y en otras perdidos. Este hecho, junto con el aumento del mercado de animales de compañía, en el que los animales son considerados como mera mercancía, ha incrementado la presión social a favor de los derechos de los animales. Esto ha provocado que surjan una gran cantidad de asociaciones encargadas de velar por los derechos de los animales e intentar mejorar la vidas de estos en la medida de lo posible.

1.1 Situación Actual

Actualmente, existen diversos mecanismos a través de los cuales obtener una mascota. El más habitual, como se ha indicado anteriormente, sigue siendo el mercado de mascotas, sin embargo, derivado de la concienciación sobre los derechos de los animales, la aparición de asociaciones que se encargan de recoger animales abandonados y darlos en adopción, ha aumentado.

Con independencia del origen de los animales, el medio mas extendido para adquirirlos en la actualidad, es Internet. Las asociaciones utilizan este canal para dar a conocer los animales que tienen en adopción encontrandoles así una vivienda con la mayor brevedad posible. Sin embargo, existen una gran cantidad de organizaciones que no pueden utilizar este sistema por los gastos que esto conlleva, porque no existen plataformas en las que diferentes asociaciones

puedan dar a conocer los animales que tienen en proceso de adopción, obligando a que creen sus propios sitios web. Por otra parte, las asociaciones que sí utilizan este medio, no alcanzan una gran visibilidad, por tratarse, en su mayor parte, de sitios web propios.

Cabe destacar también que, en muchas ocasiones, los animales que se encuentran en estas protectoras, no han sido abandonados, sino que se han extraviado. La falta de medios para que el dueño de un animal lo encuentre en caso de que esto suceda supone un problema adicional al que Internet puede dar solución.

Se hace necesario, por tanto, la existencia de una plataforma global que proporcione servicio a usuarios y asociaciones, permitiendo que:

- Diferentes asociaciones puedan dar a conocer aquellos animales que necesitan ser adoptados, mejorando así, su visibilidad y el proceso de adopción.
- Facilitar a los dueños de animales la tarea de localización de animales perdidos, lo que permitiría reducir el número de animales en protectoras.

1.2 Alcance y objetivos

El objetivo de este proyecto es el desarrollo de una aplicación que permita mostrar, en un único lugar, los animales en adopción en diferentes asociaciones, dando así una mayor visibilidad, permitiendo a los usuarios encontrar animales en adopción que cumplan con sus necesidades de una forma sencilla y rápida. Adicionalmente, la plataforma pretende ayudar en el proceso de localización de animales perdidos, favoreciendo la participación de los usuarios en dicha tarea.

La aplicación desarrollada permite el registro de usuarios y asociaciones. Por una parte, las protectoras podrán añadir animales en adopción, de forma que cualquier usuario pueda consultar, de forma rápida, los animales en adopción disponibles en las distintas asociaciones. Cualquier usuario tiene acceso al listado completo de animales en adopción y podrá consultar información detallada de estos, incluyendo su localización exacta, pudiendo contactar. de forma directa, con la asociación correspondiente. Por otra parte, un usuario registrado podrá publicar la información de un animal que se haya perdido. De esta forma, cualquier usuario que acceda a la plataforma podrá consultar animales perdidos en su zona de forma precisa, pudiendo, a su vez, indicar la posición de este, en caso de que lo haya localizado, e incluso, contactar con el dueño.

Se pretende que el desarrollo de la aplicación permita facilidad de cambios así como la inclusión de nuevas funcionalidades, construyendo así, un software más solido y robusto. Mediante la aplicación de buenas practicas y patrones de diseño se intenta facilitar el proceso de evolución del sistema, reduciendo el coste de mantenimiento.

Capítulo 2

Base Tecnológica

En las siguientes secciones se describen los lenguajes y tecnologías usadas para llevar a cabo el proyecto, además de las herramientas empleadas para dicha tarea.

2.1 Lenguajes

A continuación se describen los lenguajes utilizados para el desarrollo de la aplicación:

Java[5]: Lenguaje de propósito general y orientado a objetos. Permite el desarrollo de aplicaciones independientes de la arquitectura en la que se pretende desarrollar, gracias al JRE (Java Runtime Environment) que permite ejecutar el código fuente, ya compilado, en los diferentes entornos.

JPQL[6]: *Java Persistence Query Language*, se trata de un lenguaje de consultas utilizado por JPA (*Java Persistence API*) para obtener información de una base de datos. Permite la realización de consultas referenciando entidades del modelo de objetos de la aplicación en lugar de tablas (como es el caso de SQL).

JavaScript[7]: Se trata de un lenguaje de programación utilizado para proporcionar un comportamiento dinámico a páginas web. Se trata de un lenguaje interpretado, por lo tanto no necesita ser compilado previamente. JavaScript permite añadir una gran cantidad de características a un sitio web, como pueden ser *routing* entre diferentes componentes, conexión con servidores o seguridad. Además, permite modificar dinámicamente los elementos visuales de una aplicación.

HTML[8]: Lenguaje de marcado utilizado para determinar la estructura visual de las diferentes páginas de un sitio web.

CSS[9]: Lenguaje de diseño que permite determinar el estilo visual de los distintos elementos declarados a través de un lenguaje de marcado como HTML.

2.2 Frameworks y Librerías

A continuación se indican los *frameworks* y librerías utilizadas para desarrollar la aplicación:

SpringBoot[10]: Herramienta desarrollada para simplificar el proceso de creación de aplicaciones basadas en Spring. SprigBoot reduce la configuración necesaria para la ejecución del proyecto. Para ello autoconfigura todos los aspectos de nuestra aplicación, minimizando las tareas de este tipo que debe realizar el desarrollador.

JUnit[11]: *Framework* basado en Java que permite la realización de pruebas, unitarias y de integración, sobre los distintos componentes de un sistema en desarrollo.

React[12]: React es una librería de Javascript empleada para facilitar la construcción de interfaces de usuario interactivas. React se basa en la construcción de componentes con estado y comportamiento propio. La construcción de aplicaciones se lleva a cabo mediante componentes complejos que se comunican entre sí y cuya apariencia varía en función de su estado. Una de las características más beneficiosas de esta librería es que simplifica el proceso de reutilización de componentes de forma que se agiliza el proceso de desarrollo. Existen una gran variedad de librerías que complementan a React y que proporcionan componentes que reducen el tiempo de desarrollo. Las librerías mas importantes empleadas en el proyecto son:

- **bootstrap:[13]** Librería de HTML, CSS y JavaScript que aporta elementos visuales con estilo y comportamiento preestablecidos, de forma que agiliza el proceso de desarrollo de sitios web.
- **material design:[14]** Conjunto de pautas de diseño empleadas para el desarrollo de interfaces android.
- **reactstrap[15]:** Librería que proporciona una serie de componentes preconstruidos con elementos que proporciona bootstrap.
- **react-intl[16]:** Librería que proporciona un conjunto de componentes de React y una API que permite internacionalizar aplicaciones desarrolladas con React.

- **google-map-react[17]:** Librería de Google Maps que proporciona componentes para renderizar mapas en una aplicación desarrollada con React.
- **moment[18]**: Librería que proporciona diversos componentes que permiten la manipulación de fechas en una aplicación desarrollada con React.
- **react-router-dom[19]**: Proporciona componentes que permiten la redirección entre componentes en una aplicación desarrollada con React.
- **mdbreact[20]**: Proporciona una gran cantidad de componentes preconstruidos que utilizan Material Design y Bootstrap y permiten agilizar el proceso de diseño de interfaces web desarrolladas con React

Redux[21]: Redux es un contenedor de estado de aplicaciones que se puede utilizar en combinación con React o cualquier otro lenguaje de generación de vistas. Redux permite mantener un estado fácilmente accesible por los diferentes componentes de la aplicación, facilitando así la comunicación entre estos. Además, facilita el proceso de gestión del estado global de una aplicación.

2.3 Herramientas

A continuación se detallan las herramientas utilizadas para el desarrollo del sistema:

Apache Maven[22]: Herramienta de software utilizada para la gestión y construcción de proyectos *Java*. Proporciona un modelo de ejecución de proyectos simple, basado en un fichero XML en el que se describen las características del software a construir así como sus dependencias y el orden de construcción de sus elementos. Su motor de ejecución accede a repositorios de los que se descargan las dependencias necesarias, simplificando el proceso de construcción y ejecución del código fuente. Se basa en la plataforma de programación *JavaEE*[23] que se apoya en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. Permite gestionar proyectos software completos, desde la comprobación de que el código es correcto hasta que se despliega la aplicación, pasando por la ejecución de pruebas, generación de informes y documentación. Todo esto se lleva a cabo mediante una serie de etapas de ejecución configurables de forma sencilla.

Travis CI[24]: Sistema de integración continua que permite ejecutar y testear un proyecto de forma automática. Es posible integrarlo con un repositorio.

GitHub[25]. De esta forma verifica que los cambios que se incorporan al repositorio no producen errores en la ejecución del sistema en desarrollo.

Eclipse[26]: IDE o entorno de desarrollo integrado que permite el desarrollo de aplicaciones en diferentes lenguajes. Proporciona una serie de módulos que permiten reducir el tiempo de desarrollo entre otras funcionalidades. Permite depurar el proyecto de forma que se reduce el tiempo de localización de errores, proporciona validación del código y autocompletado y se integra con otros sistemas como *Git*. También permite incluir una gran variedad de nuevas características mediante la instalación de *plugins*.

Npm:[27]: Conjunto de herramientas para el desarrollo en *JavaScript*. Permite descargar una gran variedad de paquetes desarrollados por la comunidad y gestionar las dependencias de proyectos. Además permite automatizar tareas rutinarias de desarrollo.

DbVisualizer[28]: Herramienta gráfica que permite el acceso a la información contenida en diferentes tipos de gestores de bases de datos mediante drivers JDBC, facilitando el proceso de consulta y modificación de la información.

Postman[29]: Herramienta que permite el envío de peticiones HTTP a una API Rest. Resulta de especial utilidad para la realización de pruebas.

Git[30]: Sistema de control de versiones que facilita el proceso de seguimiento de cambios de los ficheros de un proyecto software. Se trata del software de control de versiones mas extendido.

Redmine[31] : Sistema de gestión de proyectos, que permite la distribución y organización del trabajo a realizar en un proyecto. Facilita la gestión de tiempos y proporciona información acerca del avance del proyecto.

2.4 Bases de datos

Las bases de datos empleadas durante el desarrollo y puesta en producción del sistema son:

MySQL[32]: Sistema gestor de base de datos mas extendido y utilizado por su seguridad, escalabilidad, velocidad y facilidad de instalación y uso. Se trata del sistema de Bases de datos escogido para el despliegue final de la aplicación.

H2[33]: Se trata de una base de datos relacional escrita en Java que puede ser ejecutada en cualquier sistema operativo. Su principal ventaja reside en que permite alojar toda la información en memoria consiguiendo velocidades de acceso a datos muy elevadas. La integración con *Springboot* se realiza de forma sencilla a través del *driver JDBC*. Cuenta además con documentación muy detallada.

2.5 Servidores Web

Las tecnologías empleadas para la ejecución del proyecto varían entre la fase de desarrollo y la fase de producción:

- En **desarrollo**, la aplicación servidora es ejecutada a través de *SpringBoot*, que proporciona por defecto un servidor *Tomcat* embebido. El código *JavaScript* que conforma la aplicación cliente es proporcionado por *NodeJs* que incluye un servidor de aplicaciones. Este código es descargado y ejecutado por un navegador.
- En **producción**, la parte servidora del sistema es desplegada en un servidor *Apache Tomcat*. Los archivos que conforman la capa web se encuentran disponibles para ser proporcionados por cualquier servidor que lo permita.

Capítulo 3

Estado del arte

En la actualidad, las plataformas que permiten localizar animales en estado de adopción, ofrecen una funcionalidad muy limitada, actuando, en la mayoría de los casos, como tablones de anuncios, en los que únicamente es posible leer información. En su mayor parte se tratan de páginas web pertenecientes a protectoras concretas que ofrecen un listado de los animales que se pueden adoptar en dicha asociación.

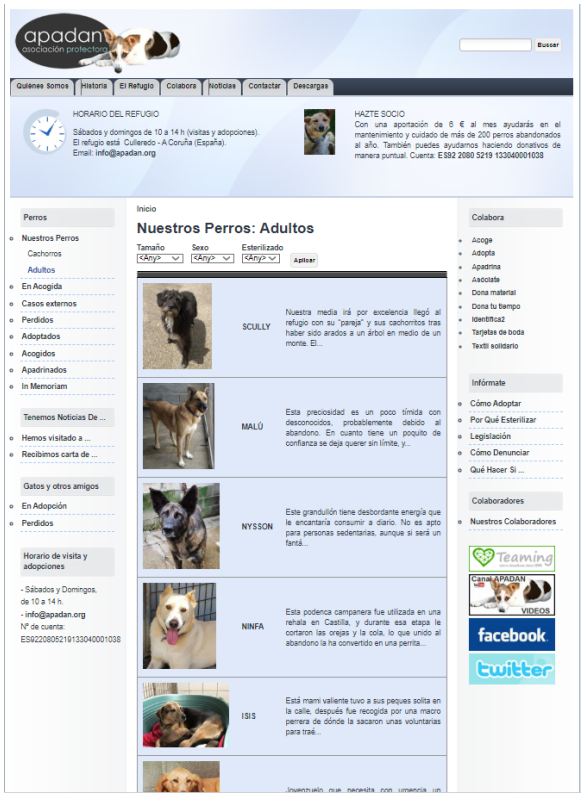


Figura 3.1: Tablón de animales en adopción de Apadan[1]

La interacción entre el usuario y estos sitios web es muy reducida. El registro de usuarios, en general, no es posible, siendo necesario, por ejemplo, el uso de herramientas externas para poner en contacto a los usuarios y protectoras, lo que dificulta la comunicación. Un ejemplo de plataforma de este tipo es *Apadan* y se muestra la página principal de su sitio web en la figura 3.1.

Existe también una gran cantidad de pequeñas asociaciones con muy pocos recursos disponibles. Estas organizaciones, por tanto, no se pueden permitir un sitio web propio. Esto provoca que recurran al uso de las redes sociales para publicitarse, lo que dificulta mucho su visibilidad, siendo difícil encontrar este tipo de perfiles. Un ejemplo de este tipo es la asociación de animales de Negreira, cuyo perfil de Facebook se muestra en la figura 3.2.

Figura 3.2: Facebook de la asociación de animales de Negreira[2]

Por otra parte, son pocas las plataformas que muestren los animales en adopción de distintas asociaciones de forma integrada. Además, los sitios que sí ofrecen este servicio a las asociaciones, no presentan funcionalidades de localización de animales cercanos, siendo necesario realizar búsquedas localidad por localidad, lo que complica el proceso. En la figura 3.3 se muestra una de las pocas plataformas que ofrecen el servicio indicado.

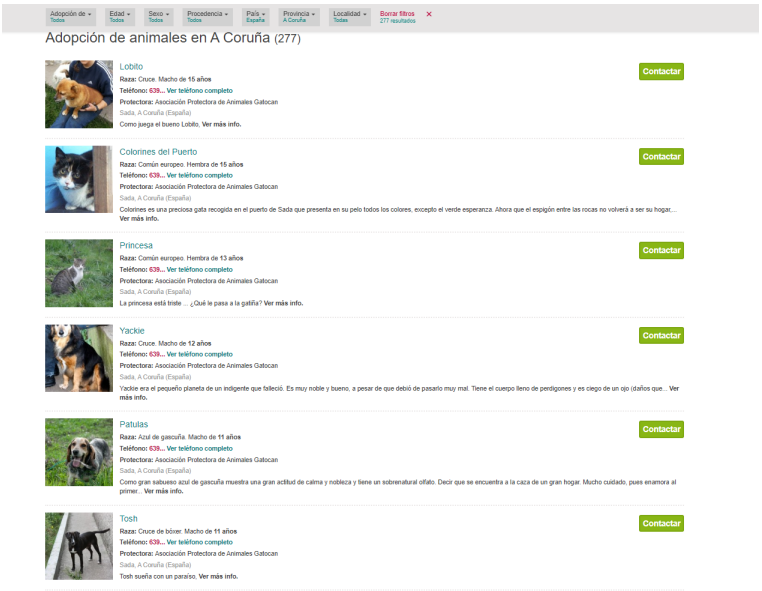


Figura 3.3: Tablón de animales en adopción de MundoAnimalia[3]

En lo que respecta a animales perdidos, la escasez de plataformas que permitan publicitar animales perdidos, provoca un aumento en el número de animales en protectoras. Los sitios que ofrecen este servicio, únicamente permiten indicar información básica acerca del animal perdido. La solución que proponen estos sitios para la localización de dichos animales es proporcionar la información de contacto de los dueños, de forma que otros usuarios se puedan poner en contacto con estos por correo electrónico o por teléfono. Como se puede ver en la figura 3.4, estas plataformas se tratan de meras páginas informativas en las que se hace necesario que los usuarios entren periódicamente para comprobar, localidad por localidad, si hay animales perdidos en su zona, ofreciendo escasa información de la posición exacta en la que el animal fue visto por última vez.

En definitiva, las plataformas que pretenden dar a conocer y facilitar el proceso de adopción y localización de animales perdidos, no proporcionan una solución muy óptima. Sus carencias más notables se pueden dividir en:

- Se tratan en su mayoría de webs de asociaciones concretas en las que la información mostrada es muy reducida.
- Dificultan el proceso de búsqueda de animales concretos que cumplan ciertas características.
- No cuentan con medios de comunicación entre usuarios.
- No proporcionan soporte para añadir o localizar animales perdidos, teniendo que recurrir a otras plataformas que no aportan una solución muy completa al problema.

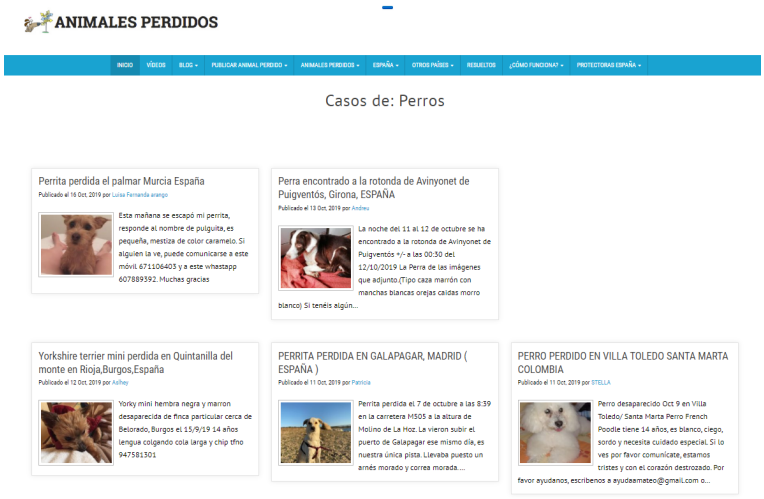


Figura 3.4: Tablón de animales perdidos animales-perdidos.org[4]

Por todo lo anterior se ha propuesto el desarrollo de la aplicación AdoptApp que pretende:

- Mostrar, en una única plataforma, animales en adopción en diferentes asociaciones, aumentando su visibilidad.
- Facilitar la tarea de búsqueda de animales en adopción que cumplan los requisitos del usuario.
- Permitir la localización de animales en adopción cercanos.
- Proporcionar un medio que permita a los usuarios añadir animales perdidos, aumentando la visibilidad y facilitando el proceso de localización, al permitir a los demás usuarios indicar la posición de estos en caso de que los hayan visto.
- Proporcionar un método propio de comunicación, de forma que usuarios y asociaciones se puedan poner en contacto sin tener que recurrir a aplicaciones externas.

Capítulo 4

Análisis de viabilidad

De forma previa a la realización de proyecto, se realiza un análisis de viabilidad para que, teniendo en cuenta los aspectos económicos, temporales y tecnológicos, y sus correspondientes restricciones, determinar si es posible su correcta realización.

En primer lugar, partiendo de la premisa de que se trata de un proyecto adaptado a la realización de un trabajo de fin de grado, el **aspecto temporal** no supone un inconveniente, puesto que, la cantidad de funcionalidades a desarrollar, se adaptan para que sea posible su realización en el tiempo establecido, suponiendo un coste en trabajo equivalente al de dos asignaturas del grado. Teniendo en cuenta lo citado, se puede determinar que la restricción de tiempo no implica un inconveniente para la correcta realización del proyecto.

Por otra parte, es necesario determinar la **disponibilidad de tecnologías** que permitan la correcta y completa realización del proyecto. En este caso, se plantea el desarrollo de una aplicación web que consta de dos partes claramente diferenciadas. Por una parte se requiere desarrollar una aplicación servidora que gestione la lógica de la aplicación, consulte y modifique la información en la base de datos y ofrezca una interfaz con las funcionalidades planteadas. Para esto, analizando diversas posibilidades, se ha llegado a la conclusión de que *SpringBoot* ofrece un solución completa al problema planteado. Por otra parte se requiere desarrollar una aplicación cliente que, utilizando la interfaz ofrecida por la parte servidor, le envíe peticiones a este, partiendo de las entradas del usuario. Existen múltiples tecnologías que permiten este desarrollo. En esta ocasión se ha determinado que *React* ofrece una buena solución, por su flexibilidad, alcance y abundancia de librerías que facilitan el desarrollo. Ambas se tratan de tecnologías ampliamente aplicadas para la construcción de aplicaciones web, por lo que no suponen mayor riesgo que el derivado de la formación en esas tecnologías. Adicionalmente, relacionado con el aspecto tecnológico, se requiere de un sistema gestor de bases de datos para almacenar la información con la que trabajará la aplicación. Se ha determinado que *mysql* es una buena opción por su fácil utilización, la abundancia de documentación y por ser uno de los más extendidos.

Por último, cabe analizar el **aspecto económico**, puesto que, al tratarse de un proyecto de fin de grado, los recursos disponibles son escasos. En este caso, los recursos necesarios son principalmente, un ordenador y acceso a Internet. La disponibilidad de ordenador propio, previa realización del proyecto, solventa el coste que implicaría obtenerlo. Por otro lado, el acceso a Internet tiene un coste muy reducido. En lo que a tecnologías se refiere, no suponen coste adicional por ser de tipo *open source*. Se puede determinar, por lo tanto, que el aspecto económico no supone un motivo para dudar de la viabilidad del proyecto.

En definitiva, analizados los principales puntos que permiten determinar la viabilidad del proyecto, se ha llegado a la conclusión que no existe ningún aspecto que impida la realización del mismo en tiempo y coste, teniendo en cuenta los recursos disponibles y las restricciones que aplican en esta ocasión.

Capítulo 5

Introducción al Desarrollo Realizado

5.1 Introducción

En este trabajo de fin de grado se propone el diseño e implementación de una aplicación web que permita mejorar los procesos de adopción de animales de compañía así como la localización y recuperación de animales perdidos.

En el capítulo anterior se han enumerado las diferentes tecnologías y herramientas utilizadas durante el desarrollo de este trabajo. En este capítulo se introduce la arquitectura de la solución indicando las tecnologías usadas en cada una de las capas. También se introduce la metodología utilizada y la división principal en iteraciones realizada.

5.2 Tecnologías

La aplicación desarrollada se basa en una arquitectura cliente-servidor. Para entender el proceso de desarrollo y encuadrar las tecnologías en cada una de las etapas, es necesario comprender la estructura de este tipo de aplicaciones.

Se dice que una aplicación sigue un modelo cliente-servidor cuando cuenta con dos partes bien diferenciadas. La función de cada una de estas partes dentro del sistema es diferente y las tecnologías empleadas para su desarrollo también pueden serlo. Se puede decir, por tanto, que el sistema se divide en dos aplicaciones. En la aplicación servidora o *backend* se establece la lógica de operaciones que realizará el sistema y se produce la conexión con la BD. Esta ofrece un interfaz que permite ser accedida por una o más aplicaciones cliente (*frontend*).

La aplicación *frontend* proporciona la interfaz de usuario. La lógica en esta parte es muy reducida. Su función principal consiste en recoger información de entrada de un usuario, enviar peticiones al backend en función de esas entradas y mostrar resultados.

El desarrollo de la aplicación cliente (*frontend*) y de la aplicación del lado servidor (*backend*) se ha llevado a cabo de forma simultanea. A continuación se describe, de forma detallada, el proceso de desarrollo de las distintas funcionalidades y las tecnologías implicadas:

Para la construcción del *backend* se ha empleado como lenguaje *Java*. Se basa en *SpringBoot*[10] y se usa *Maven*[22] para la gestión. La generación inicial del proyecto se ha llevado a cabo mediante un plugin de *Eclipse* conocido como *Spring Tools*[34], que permite la creación de un proyecto *SpringBoot* en base a un conjunto de dependencias iniciales determinado. Para el desarrollo de cada una de las funcionalidades se crean, inicialmente, las entidades necesarias. *SpringBoot* crea las tablas de base de datos automáticamente, partiendo de estas entidades. Para establecer la conexión con la BD, *Springboot* utiliza *JPA*[35] y el driver *JDBC*[36]. En tiempo de desarrollo se ha utilizado *H2*[33] como sistema de base de datos, por tratarse de una BD en memoria cuyo contenido se elimina tras cada ejecución de la aplicación. Una vez creadas las entidades es necesario determinar las operaciones que se realizarán sobre la BD. *SpringBoot* proporciona, por defecto, interfaces que permiten interactuar con la BD conocidas como DAOs (Objetos de acceso a datos). Estos permiten la realización de operaciones básicas sobre la BD. Sin embargo, ha sido necesaria la creación de funciones específicas, que realizan operaciones mas complejas sobre la BD. Estas funciones se crean dentro de los propios DAOs y las consultas que ejecutan son declaradas utilizando *JPQL*[6] . Este lenguaje permite la creación de consultas referenciando entidades en lugar de tablas de la BD. Sobre esta capa de acceso a datos se encuentra la de lógica de negocio. Esta capa se divide en diferentes servicios en función del ámbito de la funcionalidad. La aplicación *backend* proporciona una capa que gestiona la realización de peticiones a la capa de lógica de negocio. Para ello aporta una API Rest que servirá a la aplicación cliente. Esta capa se divide en una serie de controladores, que se dividen por ámbito de actuación y distribuirán las peticiones entre los distintos servicios. Las peticiones que se realizan sobre la API se llevan a cabo sobre el protocolo HTTP[37] y son del tipo GET, POST y PUT en función del tipo de petición. Cabe destacar, en último lugar, la utilización de *webSockets*[38] para la implementación de un sistema de chat nativo. Un *webSocket* es un protocolo de comunicación que permite establecer un canal de comunicación bidireccional entre un servidor y un cliente.

Por otra parte, la aplicación cliente se ha llevado a cabo utilizando *React*[12]. Se trata de un framework de *JavaScript*[7] que se basa en la construcción de componentes reutilizables que tienen su propia lógica y mantienen su propio estado. Para el desarrollo de cada una de las funcionalidades es necesario, por tanto, la creación de componentes que reciban las entradas del usuario, hagan las llamadas a la aplicación del lado servidor y muestren un resultado. Existen componentes de dos tipos: clases y constantes. Estos últimos no contienen estado. Los componentes de tipo clase extienden a *Component* y contienen una serie de información que representa su estado además de unas funciones que ejecutan cierta lógica. Un componen-

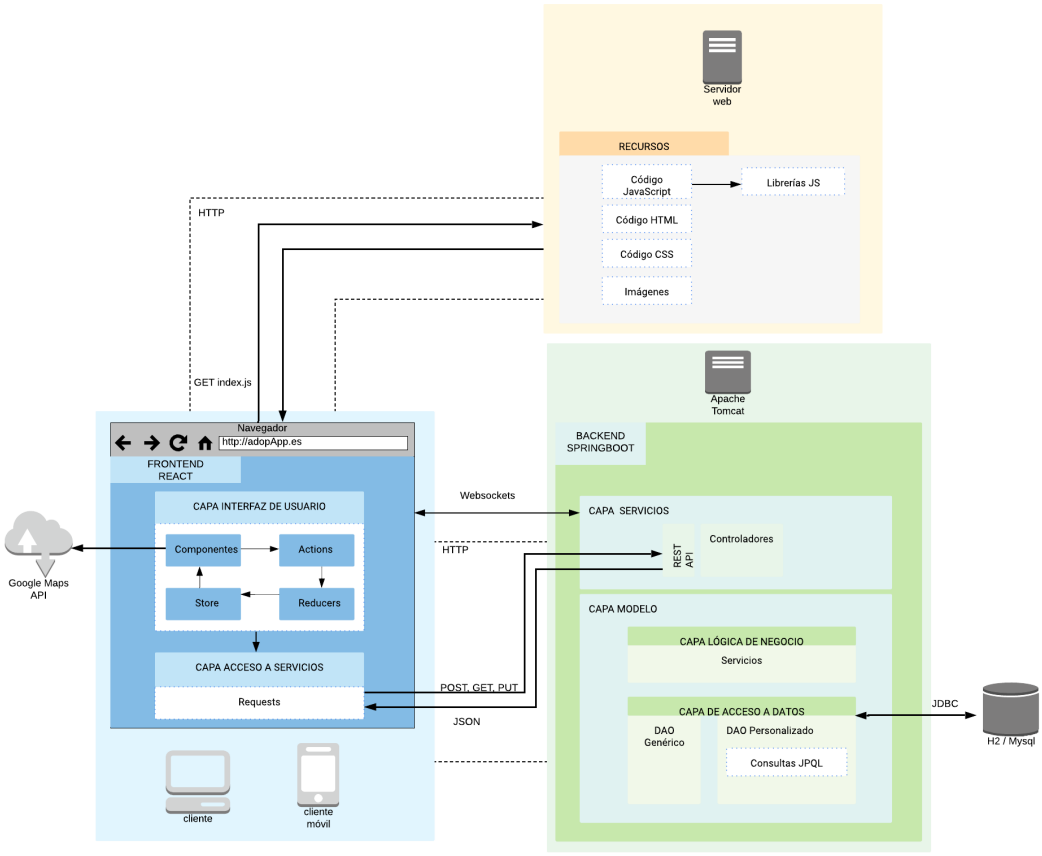


Figura 5.1: Diagrama de Arquitectura General

te de tipo clase contiene adicionalmente una función, conocida como render, que renderiza ciertos elementos a partir del lenguaje de marcado *HTML*[8] y se le aplican estilos *CSS*[9]. Se han utilizado las librerías *reactStrap*[15] y *MDBReact*[20] que proporciona componentes con estilo propio facilitando el modelaje del apartado visual. Un apartado importante de *React* y en el que se basa, es la reutilización de componentes. Dentro de una función render de un componente se puede añadir otro de forma que este se renderice dentro del principal. Como se ha dicho previamente, muchos de estos componentes realizan peticiones a la aplicación servidor (*backend*). Para esto es necesario declarar funciones que realicen estas peticiones. Más concretamente , estas funciones realizan peticiones *HTTP* de tipo POST, GET o PUT a la API REST del *backend*. En caso de que sea necesario enviar información al servidor, esta es encapsulada en formato *JSON*[39] y enviada en la petición. Cabe destacar que, en alguna ocasión, diferentes componentes necesitan compartir cierta información para operar correctamente. Para solventar este problema se ha utilizado *Redux*[21]. Este permite desacoplar el estado global de una aplicación del apartado visual, esto es, de los componentes. *Redux* uti-

liza unos elementos, conocidos como reductores, para mantener el estado compartido entre ciertos componentes. Para modificar este estado desde los componentes es necesario lanzar ciertas funciones (acciones). Una vez tratado el apartado visual y la lógica de los componentes, cabe destacar como se accede a estos y como se produce el direccionamiento. *React* incluye en *react-router-dom* un componente conocido como *Router*, que permite la asignación de componentes a rutas o paths de acceso. Cuando un usuario accede a una ruta concreta se renderiza el componente asignado a dicha ruta y sus subcomponentes. La redirección entre componentes se produce modificando esta ruta de acceso.

Una de las principales ventajas de *React* es la gran variedad de librerías y contribuciones con las que cuenta. A continuación se describen las principales utilizadas:

React-intl[16] : Se ha utilizado para internacionalizar la aplicación de forma que todos los textos incluidos se han añadido utilizando el Componente *FormattedMessage* que proporciona esta librería. De esta forma, el texto se traduce en función del idioma seleccionado.

Google-Map React[17] : Proporciona un componente que renderiza un mapa y permite la inclusión de marcadores.

Moment[18]: Permite adaptar las fechas de la aplicación a un formato concreto.

5.3 Metodologías e Iteraciones

Como marco de desarrollo se ha utilizado PUDS[40] (Proceso Unificado de desarrollo de Software). Esta metodología facilita la tarea de desarrollo de una arquitectura comprensible, que permita reutilización y cambios futuros. Este método de desarrollo se guía por casos de uso, determinando el proceso a seguir para satisfacer los requisitos del usuario final. Se trata de una metodología iterativa e incremental basada, por tanto, en el desarrollo por iteraciones de las funcionalidades del sistema.

El desarrollo de las distintas funcionalidades se ha dividido en 4 iteraciones incrementales precedidas por una de especificación de requisitos y formación. A continuación se determinan, de forma general, las tareas y funciones llevadas a cabo en cada etapa:

Iteración 0: Especificación de requisitos y formación en las tecnologías a utilizar.

Iteración 1: Registrar y modificar perfiles de usuario y de asociación.

Iteración 2: Añadir, modificar y consultar animales en adopción.

Iteración 3: Añadir, modificar,consultar y localizar animales perdidos.

Iteración 4: Añadir chat nativo, boletines semanales de información, sistema de notificaciones y publicación en redes sociales.

En el la sección 6 se analizan más en detalle cada una de las iteraciones.

Capítulo 6

Planificación y Analisis de Costes

Para la realización del proyecto se ha empleado la metodología PUDS (Proceso Unificado de Desarrollo de Software). Según la aplicación de este modelo, se ha dividido el desarrollo de las funcionalidades en cuatro etapas o iteraciones precedidas de una adicional de especificación de requisitos y formación. La división del trabajo se ha llevado a cabo de tal forma que cada una de las iteraciones cuenta con tiempo estimado similar. En las primeras iteraciones se llevan a cabo las funcionalidades más relevantes, dejando las de menor peso para las últimas etapas. El proceso de desarrollo en cada etapa consiste en: Análisis de las funcionalidades a desarrollar, implementación y pruebas. Tras finalizar cada iteración se lleva a cabo una reunión con el jefe de proyecto para analizar las funciones desarrolladas y el progreso del proyecto. A continuación se detalla cada iteración:

Iteración 0: Se establecen los requisitos del sistema y las funcionalidades a desarrollar. Además se lleva a cabo un periodo de formación en las diferentes tecnologías y herramientas necesarias para el desarrollo.

Iteración 1: Se genera el proyecto *SpringBoot* base. Se realiza la configuración básica para que utilice H2 como BD. En esta etapa las funcionalidades desarrolladas son principalmente la creación y modificación de perfiles de usuario y asociación. Las tareas concretas llevadas a cabo en esta iteración se pueden ver en la figura 6.1

Iteración 2: Como se pueden ver en la figura 6.2, en la segunda iteración las tareas llevadas a cabo permiten que los perfiles de asociación puedan añadir animales en adopción. Estas tienen acceso al listado de animales que han añadido y pueden borrarlos o modificar su información. Cualquier usuario autenticado o no, tiene acceso al listado de animales en adopción. En caso de que el usuario esté registrado, estos se muestran ordenados por proximidad. Los usuarios pueden además, acceder a la información detallada del animal. Existe, por otra parte, la posibilidad de visualizar un mapa con la posición de las asociaciones, pudiendo ver el listado de animales disponibles en esta.

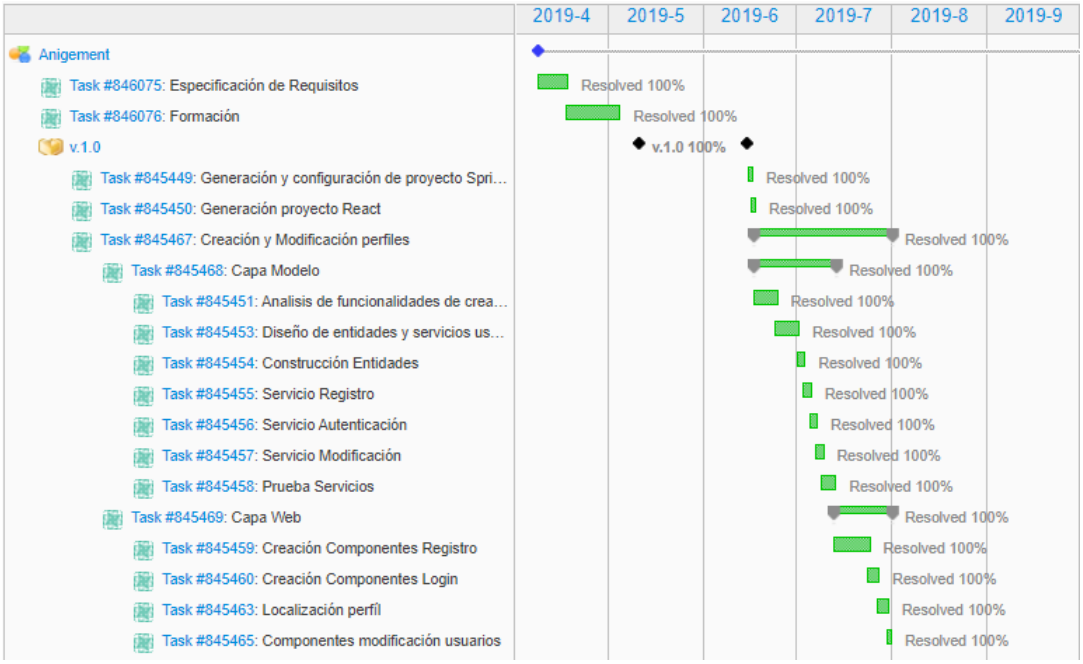


Figura 6.1: Planificación de la iteración 1

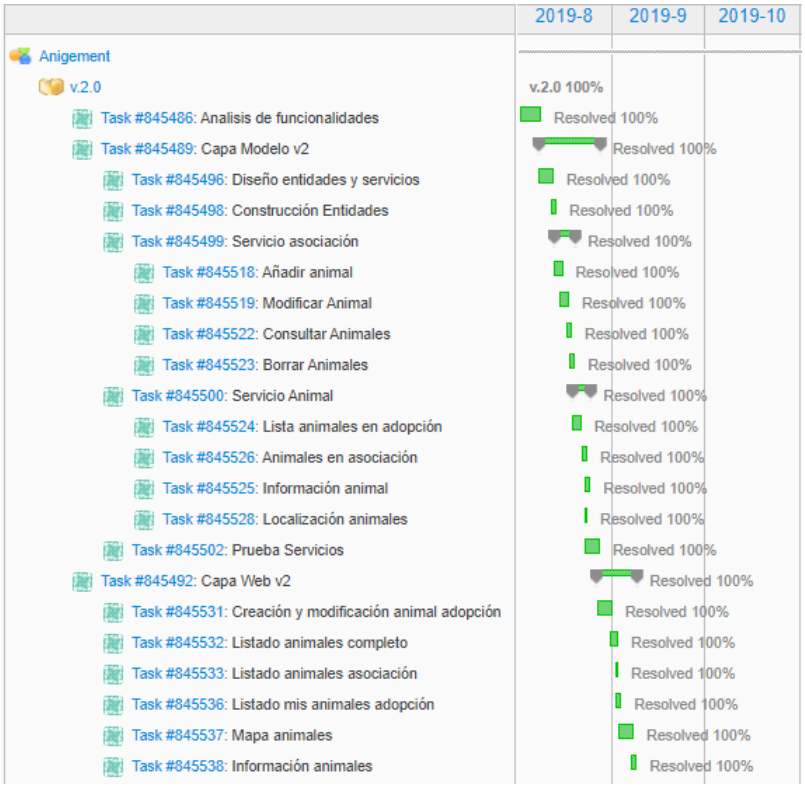


Figura 6.2: Planificación de la iteración 2

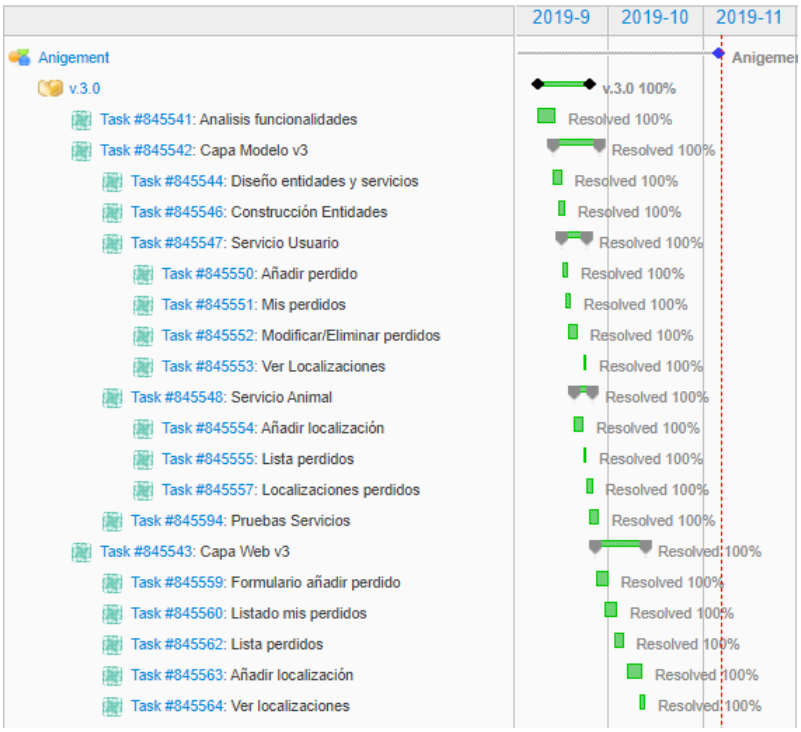


Figura 6.3: Planificación de la iteración 3

Iteración 3: Los usuarios registrados pueden añadir animales perdidos a la aplicación, tienen acceso al listado de animales que hayan añadido y pueden modificar su información o borrarlos. Cualquier usuario tiene acceso al listado completo de animales perdidos y puede indicar la posición en la que lo ha visto. También se ha desarrollado un mapa en que se pueden ver los animales perdidos por localización. El dueño de un animal tiene acceso a un mapa con las localizaciones en las que cada uno de sus animales ha sido visto. En la figura 6.3 se pueden ver las tareas llevada a cabo para desarrollar esta funcionalidad.

Iteración 4: Tanto usuarios registrados como asociaciones pueden ponerse en contacto a través de un chat nativo. Estos tienen acceso a un listado de chats activos y pueden acceder a ellos para consultar mensajes enviados previamente o enviar nuevos. Cuando una asociación añada un animal, los usuarios cuyas preferencias coincidan con las características de este serán notificados. Cuando se añada un animal perdido, los usuarios cercanos serán notificados de igual forma. Todo usuario registrado tiene acceso a un listado de notificaciones y puede borrarlas una vez las haya visto. Tanto los usuarios registrados como las asociaciones pueden indicar en sus preferencias una distancia máxima en la que se muestran animales en adopción y perdidos en los mapas de la aplicación. Además solo recibirán notificaciones de animales en esa zona. Los usuarios pueden indicar unas preferencias de animal, de forma que sean notificados cuando se añada un animal en adopción que cumpla con las características

especificadas en dichas preferencias. Los usuarios recibirán semanalmente un boletín en su correo electrónico con los animales que cumplen con sus preferencias. Por último, cuando una asociación añade un animal, cuenta con la posibilidad de compartirlo en sus redes sociales. La planificación de esta iteración se puede ver en la figura 6.4

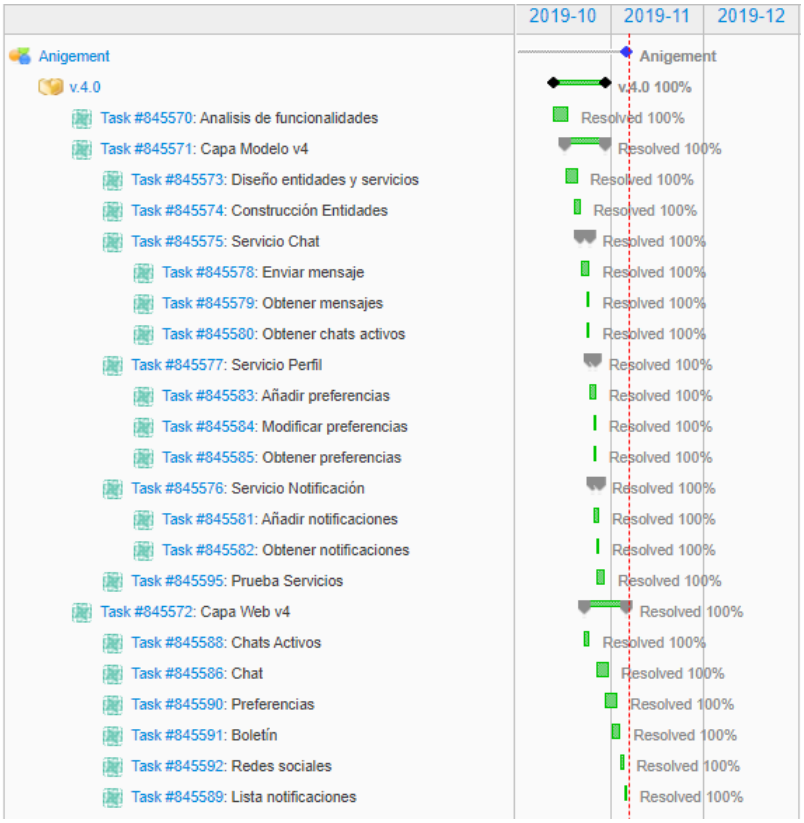


Figura 6.4: Planificación de la iteración 4

La planificación establecida inicialmente se ha cumplido en términos de tareas por iteración. Sin embargo, en términos de tiempo, el proceso de desarrollo se ha alargado 12 días en la tercera iteración debido a la conciliación con la actividad laboral lo que a reducido el porcentaje de dedicación al proyecto. A continuación se detalla el coste por iteración en término de horas reales.

Etapa	**Iteración**	**Coste**
Iteración 0	Analista	50 horas
Iteración 1	Analista	45 horas
	Programador	91 horas
Iteración 2	Analista	42 horas
	Programador	80 horas
Iteración 3	Analista	39 horas
	Programador	82 horas
Iteración 4	Analista	30 horas
	Programador	76 horas
Coste Total		535 horas

A partir del coste temporal invertido en el proyecto y suponiendo un coste por hora del programador de 13 euros/hora y de 18 euros/hora para el analista. El coste económico del trabajo realizado se detalla a continuación:

Etapa	**iteración**	**Coste**
Iteración 0	Analista	900 euros
Iteración 1	Analista	810 euros
	Programador	1183 euros
Iteración 2	Analista	756 euros
	Programador	1040 euros
Iteración 3	Analista	702 euros
	Programador	1066 euros
Iteración 4	Analista	540 euros
	Programador	988 euros
Coste Total		6919 euros

Además es necesario considerar gastos adicionales derivados de la infraestructura básica. En la siguiente tabla se pueden ver asociados a su coste.

Elemento	**Coste**
Electricidad	120 euros
Equipos informáticos	60 euros
Internet	240 euros
Coste Total	240 euros

Podemos determinar, por lo tanto, un coste económico total para el proyecto de **7.159 euros**.

Capítulo 7

Requisitos del sistema

7.1 Introducción

La aplicación desarrollada pretende dar servicio a diferentes asociaciones de animales, facilitando la tarea de darse a conocer y aumentar la visibilidad de los animales que tienen en proceso de adopción. Por otra parte, permite a los usuarios encontrar animales perdidos en el menor tiempo posible, proporcionando funciones que permiten publicitarlos así como potenciar la participación de otros usuarios en la tarea de encontrarlos. A continuación se describen los requisitos de la aplicación:

Las asociaciones que se registren pueden acceder al sistema para añadir animales indicando información sobre estos, eliminar animales de las listas de adopción y editar la información de los animales. Al añadir un animal en adopción , la asociación puede publicarlo en redes sociales. Los usuarios que se registren en la aplicación tienen acceso a un listado de animales en adopción en las distintas asociaciones, ordenados por proximidad (si el usuario ha configurado su posición), permitiendo realizar filtrados sobre este listado. También se tiene acceso al listado de asociaciones. Los listados muestran la información de cada animal y asociación y los usuarios pueden usar un chat nativo para ponerse en contacto con estas. Adicionalmente, se tiene acceso a un mapa en el que se pueden visualizar los animales en adopción en función de su posición. Un usuario que no esté registrado también tiene acceso a estos listados, sin embargo, solo un usuario registrado se puede poner en contacto con una asociación.

Los usuarios tienen acceso a la configuración de su cuenta donde pueden configurar las características que esperan de un animal que pretendan adoptar. De esta forma el sistema notificará al usuario cada vez que se añada un animal a la lista de adopción que cumpla sus necesidades. Adicionalmente, los usuarios recibirán semanalmente un boletín en su correo electrónico (si así lo desean) con el listado de animales añadidos esa semana que cumplan con las características especificadas.

Otro conjunto de funciones importantes con las que cuenta la aplicación están orientadas a

localizar animales perdidos. Un usuario puede añadir un anuncio en caso de que haya perdido un animal. Este indicará la información sobre el animal, además de la localización donde se haya perdido. Todos los usuarios que han indicado en su perfil una localización recibirán una notificación cuando se haya perdido un animal en su zona. Cualquier usuario tiene acceso al listado de animales perdidos además de a un mapa con la posición exacta de estos. Los usuarios se pueden poner en contacto con el dueño del animal perdido en caso de que lo haya encontrado o puede indicar en un mapa la posición en la que lo ha visto. Esto notifica al usuario que ha perdido al animal y tiene acceso a un mapa con la ruta que ha seguido el animal, mostrada a través de los puntos marcados por los distintos usuarios de la aplicación.

La aplicación está adaptada para poder ser visualizada y usada a través de diferentes tipos de dispositivos y pantallas, potenciando así, el uso por parte de un mayor número de usuarios.

7.2 Actores

A continuación se indican los distintos actores que interactúan con el sistema:

Usuario Anónimo: Se trata de cualquier usuario que acceda al sistema sin identificarse, es decir, sin pasar por el proceso de autenticación.

Usuario Registrado: Se trata de cualquier usuario que acceda al sistema pasando por el proceso de autenticación, independientemente del rol.

Usuario Individual: Se tratan de usuarios autenticados con rol de usuario. Estos actores tienen acceso a las mismas funcionalidades que un usuario anónimo y además pueden añadir animales perdidos y ponerse en contacto con otros usuarios. Además pueden recibir boletines de información sobre nuevos animales en adopción añadidos.

Usuario Asociación: Se trata de un rol de un usuario autenticado por lo que tiene acceso a las mismas funcionalidades a las que pueden acceder estos. Además tiene las funcionalidades de un usuario anónimo. Adicionalmente pueden añadir animales en adopción y gestionarlos.

La relación entre los distintos tipos de actores se puede ver en la figura 7.1

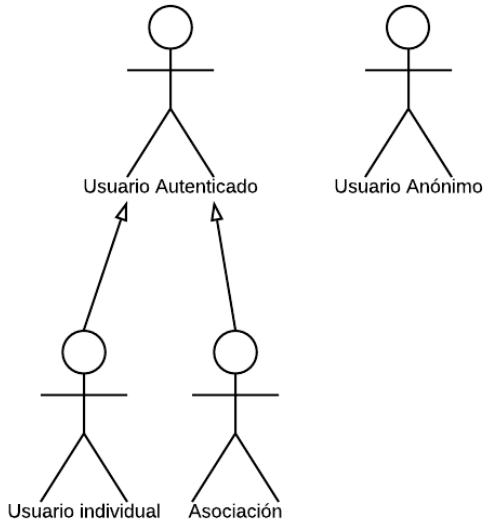


Figura 7.1: Jerarquía de actores

7.3 Casos de uso

A continuación se definen los casos de uso de la aplicación:

CU-01	**Registrar perfil**
Precondiciones	El usuario no está autenticado ni registrado
Flujo normal	El usuario introduce los campos requeridos y pulsa el botón de *Registrase*
Post-Condiciones	Se almacena la información del usuario en el sistema y este accede a la aplicación como usuario registrado
Excepciones	El usuario no introduce la información obligatoria, el nombre de usuario ya existe o el correo electrónico no tiene el formato correcto
CU-02	**Iniciar sesión**
Precondiciones	El usuario está registrado pero no autenticado
Flujo normal	El usuario introduce el nombre y la contraseña y pulsa el botón de *Acceder*
Post-Condiciones	El usuario accede a la aplicación como usuario registrado
Excepciones	El usuario no introduce el nombre o la contraseña o alguno de los valores es inválido

CU-03	**Cerrar Sesión**
Precondiciones	Estar autenticado
Flujo normal	El usuario selecciona la opción *Perfil - Cerrar Sesión*
Post-Condiciones	El usuario deja de estar autenticado y es redirigido a la pantalla de login
Excepciones	Ninguna
CU-04	**Indicar localización**
Precondiciones	El usuario se ha registrado satisfactoriamente
Flujo normal	El usuario indica su posición en el mapa y pulsa *Añadir localización*
Post-Condiciones	La posición del usuario se almacena en el sistema y se accede a la aplicación como usuario registrado
Excepciones	El usuario pulsa *Añadir localización* sin haber indicado una posición en el mapa
CU-05	**Consultar información de perfil**
Precondiciones	El usuario esta autenticado
Flujo normal	El usuario selecciona la opción *Perfil - Información Personal* en el menú. Se muestra la información del perfil.
Post-Condiciones	Ninguna
Excepciones	Ninguna
CU-06	**Modificar información de perfil**
Precondiciones	El usuario esta autenticado y ha accedido a la página de *Información Personal* (CU-05).
Flujo normal	El usuario modifica la información que desee y pulsa *Actualizar Información.* Aparece un mensaje indicando que la información se ha modificado satisfactoriamente.
Post-Condiciones	La nueva información se almacena en el sistema.
Excepciones	El usuario pulsa *Actualizar información* sin haber introducido todos los campos obligatorios o el correo electrónico no tiene un formato válido.

CU-07	**Añadir animal en adopción**
Precondiciones	El usuario esta autenticado con perfil de asociación.
Flujo normal	Se selecciona la opción *Animales adopción - Añadir animal* del menú. Aparece un formulario en el que el usuario rellena la información del animal y pulsa el botón *Añadir*. Aparece un mensaje que indica que el animal se ha añadido correctamente.
Post-Condiciones	La información del animal se almacena en el sistema.
Excepciones	El usuario pulsa el botón *Añadir* sin haber rellenado algún campo obligatorio.
CU-8	**Modificar animal en adopción**
Precondiciones	El usuario esta autenticado con perfil de asociación y ha accedido a la página de consulta de animales en asociación (CU-22).
Flujo normal	El usuario pulsa el botón *Editar* de alguno de los animales disponibles en la asociación. Aparece un formulario con la información actual del animal. El usuario modifica la información que considere y pulsa el botón *Actualizar*. Aparece un mensaje que indica que la información se ha actualizado correctamente.
Post-Condiciones	La información del animal se actualiza en el sistema.
Excepciones	El usuario pulsa el botón *Actualizar* sin haber rellenado todos los campos obligatorios.
CU-9	**Eliminar animal en adopción**
Precondiciones	El usuario esta autenticado con perfil de asociación y ha accedido a la página de consulta de animales en asociación (CU-22).
Flujo normal	El usuario pulsa el botón *Borrar* de alguno de los animales disponibles.
Post-Condiciones	La información del animal se elimina del sistema. La lista de animales en la asociación se actualiza.
Excepciones	Ninguna.

CU-10	**Ver información de animal en adopción**
Precondiciones	El usuario ha accedido a la lista completa de animales en adopción (CU-10) o a la lista de animales en adopción en una asociación (CU-21).
Flujo normal	El usuario pulsa el botón *Más Información* de alguno de los animales disponibles. Se muestra una página con información detallada del animal, incluyendo fotografías de este y su localización.
Post-Condiciones	Ninguna
Excepciones	El animal no existe
CU-11	**Ver listado completo de animales en adopción**
Precondiciones	El usuario ha accedido al sistema de forma anónima o se ha autenticado.
Flujo normal	El usuario selecciona la opción *Animales Adopción - Lista de Animales* del menú. Se muestra el listado completo de animales en adopción en las distintas asociaciones.
Post-Condiciones	Ninguna
Excepciones	Ninguna
CU-12	**Creación de animal perdido**
Precondiciones	El usuario ha accedido al sistema con perfil de usuario.
Flujo normal	El usuario selecciona la opción *Animales Perdidos - Añadir Animal* del menú. Se muestra un formulario en el que el usuario introduce la información del animal y pulsa el botón *Añadir Animal.*
Post-Condiciones	La información del animal se almacena en el sistema y el usuario es redirigido al listado de sus animales perdidos (CU-25).
Excepciones	El usuario pulsa el botón *Añadir animal* sin haber rellenado alguno de los campos obligatorios del formulario.

CU-13	**Modificar animal perdido**
Precondiciones	El usuario se ha autenticado con perfil de usuario y ha accedido a la lista de sus animales perdidos (CU-23).
Flujo normal	El usuario pulsa el botón *Editar* de alguno de los animales perdidos disponibles. Se muestra un formulario con la información actual del animal. El usuario actualiza la información que considere y pulsa el botón *Actualizar información*. Se muestra un mensaje que indica que la información se ha actualizado correctamente.
Post-Condiciones	La información actualizada se almacena en el sistema.
Excepciones	El usuario pulsa el botón *Actualizar Información* sin haber rellenado todos los campos obligatorios del formulario.
CU-14	**Eliminar animal perdido**
Precondiciones	El usuario se ha autenticado con perfil de usuario y ha accedido a la lista de sus animales perdidos (CU-23).
Flujo normal	El usuario pulsa el botón *Borrar* de alguno de los animales perdidos disponibles.
Post-Condiciones	La información del animal se borra del sistema y el listado de animales perdidos del usuario se actualiza.
Excepciones	Ninguna
CU-15	**Ver información de animal perdido**
Precondiciones	Haber accedido al listado completo de animales perdidos (CU-16).
Flujo normal	El usuario pulsa el botón de *Mas Información* de alguno de los animales disponibles. Se muestra una página con los detalles del animal. incluyendo sus fotografías y la posición exacta donde el dueño lo vio por ultima vez.
Post-Condiciones	Ninguna
Excepciones	El animal no existe

CU-16	**Ver listado completo de animales perdidos**
Precondiciones	Haber accedido al sistema como usuario anónimo o usuario registrado.
Flujo normal	El usuario selecciona la opción *Animales perdidos - Lista de animales* del menú. Se muestra el listado completo de animales perdidos incluyendo, para cada uno, una fotografía y una breve descripción de este.
Post-Condiciones	Ninguna
Excepciones	Ninguna
CU-17	**Ver localización de asociaciones**
Precondiciones	Haber accedido al sistema como usuario anónimo o usuario registrado
Flujo normal	El usuario selecciona la opción *Animales Adopción - Mapa asociaciones* del menú. Se muestra un mapa que incluye las localizaciones de las distintas asociaciones.
Post-Condiciones	Ninguna
Excepciones	Ninguna
CU-18	**Ver información de asociación**
Precondiciones	Haber accedido a la página de localización de asociaciones (CU-17)
Flujo normal	El usuario selecciona uno de los marcadores disponibles en el mapa. Aparece un modal con información de la asociación seleccionada.
Post-Condiciones	Ninguna
Excepciones	Ninguna
CU-19	**Ver localización de animales perdidos**
Precondiciones	Haber accedido al sistema como usuario anónimo o usuario registrado
Flujo normal	El usuario selecciona la opción *Animales Perdidos - Mapa de animales* del menú. Se muestra un mapa que incluye las localizaciones en las que se han perdido animales.
Post-Condiciones	Ninguna
Excepciones	El animal no existe

CU-20	**Ver listado de asociaciones**
Precondiciones	Haber accedido al sistema como usuario anónimo o registrado
Flujo normal	El usuario selecciona la opción *Animales Adopción - Lista Asociaciones* del menú. Se muestra una lista de las asociaciones disponibles incluyendo, para cada una, una breve descripción.
Post-Condiciones	Ninguna
Excepciones	Ninguna
CU-21	**Ver animales en adopción de asociación**
Precondiciones	El usuario ha accedido al listado de asociaciones (CU-20) o al mapa de localización de asociaciones(CU-19)
Flujo normal	El usuario pulsa el botón *Ver animales* de alguno de los animales disponibles en la lista. Se muestra el listado de animales disponibles en la asociación. Para cada uno de ellos se muestra una fotografía además de una breve descripción.
Flujo alternativo	El usuario selecciona alguno de los marcadores del mapa, correspondientes a la localización de cada una de las asociaciones, se muestra un modal con información de la asociación. El usuario pulsa el botón *Ver animales* disponible en el modal. Se muestra el listado completo de animales disponibles en la asociación. Para cada uno de ellos se muestra una fotografía además de una breve descripción.
Post-Condiciones	Ninguna
Excepciones	La asociación no existe
CU-22	**Ver animales en asociación**
Precondiciones	Estar autenticado con perfil de asociación
Flujo normal	El usuario selecciona la opción *Animales Adopción - Mis animales*. Se muestra el listado de animales disponibles en la asociación, en estado de adopción o que ya han sido adoptados. Para cada animal se muestra una fotografía y una breve descripción.
Post-Condiciones	Ninguna
Excepciones	Ninguna

CU-23	**Ver animales perdidos de usuario**
Precondiciones	Estar autenticado con perfil de usuario
Flujo normal	El usuario selecciona la opción *Animales Perdidos - Mis Animales*. Se muestra un listado con los animales perdidos añadidos por el usuario. Para cada animal se muestra una fotografía y una breve descripción..
Post-Condiciones	Ninguna
Excepciones	Ninguna
CU-24	**Iniciar chat**
Precondiciones	Estar autenticado y haber accedido a la pagina de información de algún animal perdido(CU-15).
Flujo normal	El usuario pulsa el botón *Contactar*. Se muestra un chat que contiene los mensajes que ambos usuarios se hayan intercambiado previamente.
Post-Condiciones	Si es la primera vez que ambos usuarios contactan, la información del chat se almacena en el sistema
Excepciones	El usuario no existe
CU-25	**Enviar mensaje en el chat**
Precondiciones	Estar autenticado y haber accedido a la página de chat con un usuario(CU-24) o con una asociación(CU-26).
Flujo normal	El usuario escribe un mensaje en el chat y pulsa Enter para enviarlo. El mensaje se muestra en la ventana del chat indicando el usuario que lo ha enviado.
Post-Condiciones	El mensaje se almacena en el sistema y el mensaje es notificado a su destinatario
Excepciones	Ninguna
CU-26	**Ver chats activos**
Precondiciones	Estar autenticado
Flujo normal	El usuario selecciona la opción *Chats Activos*. Se muestra un listado con los chats activos que tiene el usuario. Para cada chat se muestra el usuario con el que se ha contactado y es posible acceder a dicho chat.
Post-Condiciones	Ninguna
Excepciones	Ninguna

CU-27	**Acceder a chat activo**
Precondiciones	Estar autenticado y haber accedido a la página de chats activos (CU-26)
Flujo normal	El usuario pulsa el botón *Acceder* de alguno de los chats que tiene disponibles. Se muestra el chat con los mensajes previamente intercambiados entre los usuarios y es posible enviar más mensajes.
Post-Condiciones	Ninguna
Excepciones	El usuario no existe
CU-28	**Localizar animal perdido**
Precondiciones	Haber accedido al sistema como usuario anónimo o registrado y haber entrado en el listado completo de animales perdidos (CU-16)
Flujo normal	El usuario pulsa el botón *Localizar* de alguno de los animales disponibles. Se muestra un modal en el que el usuario introduce la localización en la que ha visto al animal, su nombre y un comentario si lo desea. El usuario pulsa el botón *Añadir*
Post-Condiciones	La localización se almacena en el sistema
Excepciones	El usuario pulsa el botón *Añadir* sin haber introducido una localización.
CU-29	**Ver localizaciones de un animal perdido**
Precondiciones	Estar autenticado con perfil de usuario y haber accedido a su lista de animales perdidos (CU-23)
Flujo normal	El usuario pulsa el botón *Localizaciones* de alguno de los animales disponibles. Se muestra un mapa en el que aparece un marcador por cada localización en la que un usuario ha visto al animal.
Post-Condiciones	Ninguna
Excepciones	Ninguna

CU-30	**Ver información de una localización**
Precondiciones	Estar autenticado con perfil de usuario y haber accedido al mapa de localizaciones de un animal perdido (CU-29)
Flujo normal	El usuario selecciona el marcador correspondiente a una localización concreta. Se muestra un modal con información relativa al usuario que ha añadido la localización así como un comentario sobre esta.
Post-Condiciones	Ninguna
Excepciones	Ninguna
CU-31	**Consultar preferencias**
Precondiciones	Estar autenticado
Flujo normal	El usuario selecciona la opción *Perfil - Preferencias* del menú. Se muestra una página con las preferencias del usuario.
Post-Condiciones	Ninguna
Excepciones	Ninguna
CU-32	**Modificar preferencias de distancia**
Precondiciones	Estar autenticado y haber accedido a la página de preferencias de usuario (CU-31)
Flujo normal	El usuario modifica las preferencias de distancia que considere y pulsa el botón *Actualizar información*. Se muestra un mensaje indicando que la información se ha modificado correctamente.
Post-Condiciones	La nueva información se almacena en el sistema
Excepciones	El usuario pulsa el botón *Actualizar información* sin haber rellenado alguno de los campos obligatorios
CU-33	**Modificar preferencias de animal**
Precondiciones	Estar autenticado con perfil de usuario y haber accedido a la página de preferencias (CU-31)
Flujo normal	El usuario modifica las preferencias de animal y pulsa el botón *Actualizar información*. Se muestra un mensaje indicando que la información se ha modificado correctamente.
Post-Condiciones	La nueva información se almacena en el sistema
Excepciones	El usuario pulsa el botón *Actualizar información* sin haber rellenado alguno de los campos obligatorios

CU-34	**Compartir animal en redes sociales**
Precondiciones	El usuario esta autenticado con perfil de asociación y ha accedido a sus lista de animales(CU-22)
Flujo normal	El usuario selecciona el icono de Twitter de alguno de los animales disponibles. Se muestra una ventana en la que añadir la publicación. El usuario introduce la información que desee y pulsa el botón *Publicar*
Post-Condiciones	La información se publica en Twitter
Excepciones	Ninguna
CU-35	**Consultar notificaciones**
Precondiciones	Estar autenticado con perfil de usuario y haber recibido notificaciones
Flujo normal	El usuario selecciona la opción *Notificaciones* del menú. Se muestra un listado con las notificaciones del usuario, indicando para cada una de estas el contenido de la notificación y su fecha y hora.
Post-Condiciones	Ninguna
Excepciones	Ninguna
CU-36	**Borrar Notificación**
Precondiciones	Estar autenticado con perfil de usuario y acceder a la página de consulta de notificaciones (CU-35)
Flujo normal	El usuario pulsa el botón *Borrar* de alguna de las notificaciones disponibles. .
Post-Condiciones	La información de la notificación se borra del sistema. El listado de notificaciones se actualiza.
Excepciones	Ninguna

7.4 Modelo de casos de uso

Para comprender más en detalle la relación entre las diferentes funciones del sistema y los actores que las ejecutan, a continuación se presentan diferentes modelos de caso de uso.

En la figura 7.2 se muestra la relación entre las funciones de gestión de perfiles y los actores que las llevan a cabo. Como se puede ver, únicamente se puede registrar un usuario anónimo. Un usuario registrado, por otra parte, puede iniciar sesión con sus credenciales o cerrarla si se encuentra autenticado. Cualquier usuario registrado puede acceder a su información de perfil y modificarla, incluyendo la localización. Estos pueden modificar sus preferencias de

distancia, esto es, la distancia máxima en la que se muestran animales perdidos y en adopción en los diferentes mapas. Por último, un usuario registrado puede consultar sus notificaciones y borrarlas si así lo desea.

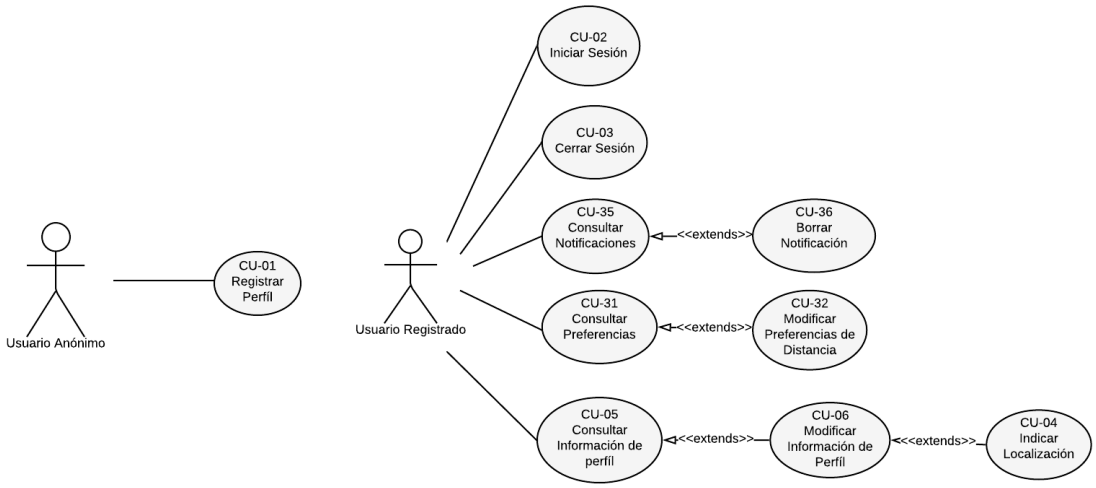


Figura 7.2: Modelo de casos de uso de gestión de perfiles

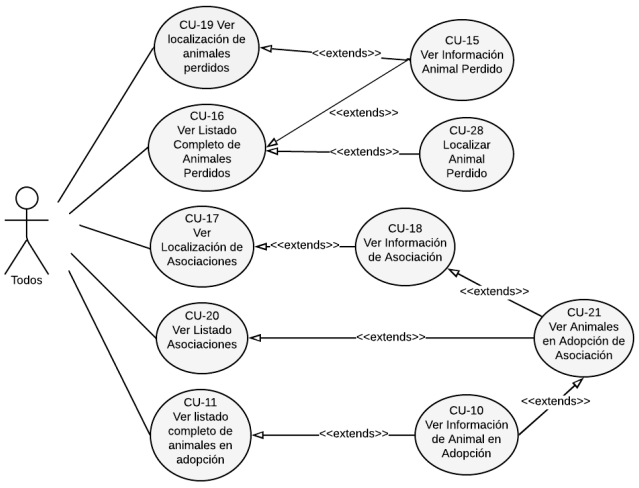


Figura 7.3: Modelo de casos de uso comunes entre los actores

Como se puede ver en la figura 7.3 la información sobre animales perdidos y animales en adopción es accesible por cualquier usuario. Cualquier persona, esté autenticada o no, tiene acceso al listado de animales en adopción y al de animales perdidos. Estos usuarios pueden ver la posición de estos animales en los mapas correspondientes. Para cada animal, el usuario puede consultar información detallada de este. Es posible además consultar el listado de

asociaciones y acceder a su lista de animales en adopción. Cualquier usuario podrá, además, indicar la localización de un animal perdido si lo ha visto.

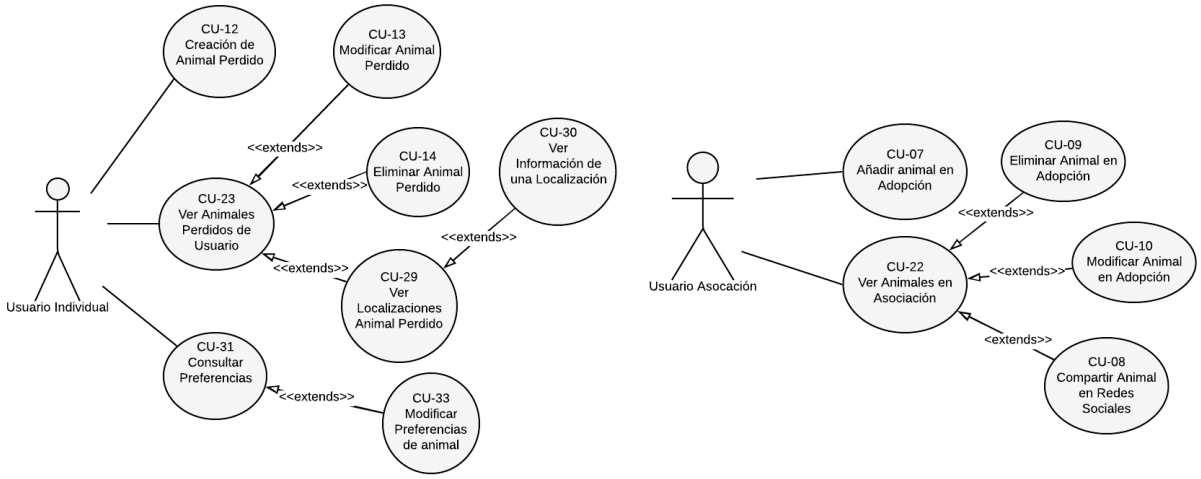


Figura 7.4: Modelo de casos de uso de usuarios y asociaciones

Los usuarios registrados con perfil de asociación se encargan de gestionar los animales en adopción. Como se pueden ver en la figura 7.4, estos pueden añadir un animal, modificarlo y eliminarlo. Pueden además añadir una publicación en redes sociales, indicando que han añadido dicho animal. Por otra parte, los demás usuarios registrados pueden añadir animales perdidos, editarlos y borrarlos. Además pueden visualizar los puntos del mapa en los que otros usuarios han visto a los animales que este ha añadido. A diferencia de las asociaciones que únicamente pueden editar las preferencias de distancia, un usuario puede editar sus preferencias de animal, esto es, las características que el usuario requiere que cumpla un animal en adopción. De esta forma el usuario será notificado cuando se añada un animal que cumpla dichas características, además de recibir un boletín semanal en su correo electrónico con estos animales.

Por último, en la figura 7.5 se pueden ver los diferentes casos de uso involucrados en la comunicación entre usuarios. Es necesario autenticarse para poder ejecutar estas funciones. Un usuario o una asociación puede iniciar un chat desde la página de información de un animal en adopción o perdido. Una vez creado el chat los usuarios se pueden enviar mensajes mutuamente.

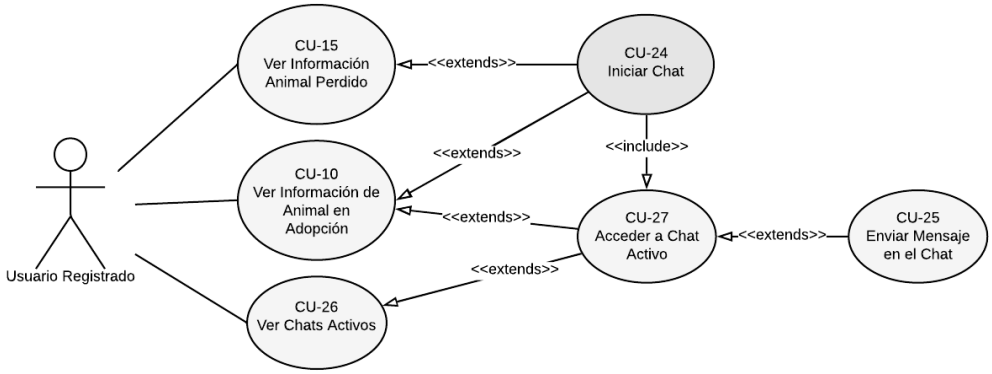


Figura 7.5: Modelo de casos de uso de la funcionalidad de chat

Para llevar a cabo el desarrollo de las funcionalidades citadas, se han creado previamente unos prototipos de pantallas de la aplicación que permiten determinar el diseño general de esta y asegurar el cumplimiento de los requisitos.

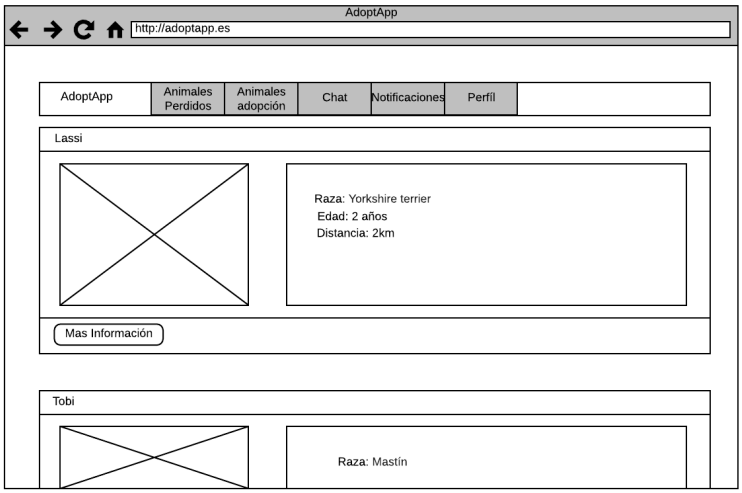


Figura 7.6: Mockup de listas

Al acceder a la aplicación, el usuario tendrá acceso al lista de animales en adopción o al de animales perdidos. Estas listas pueden verse representadas en la figura 7.6.

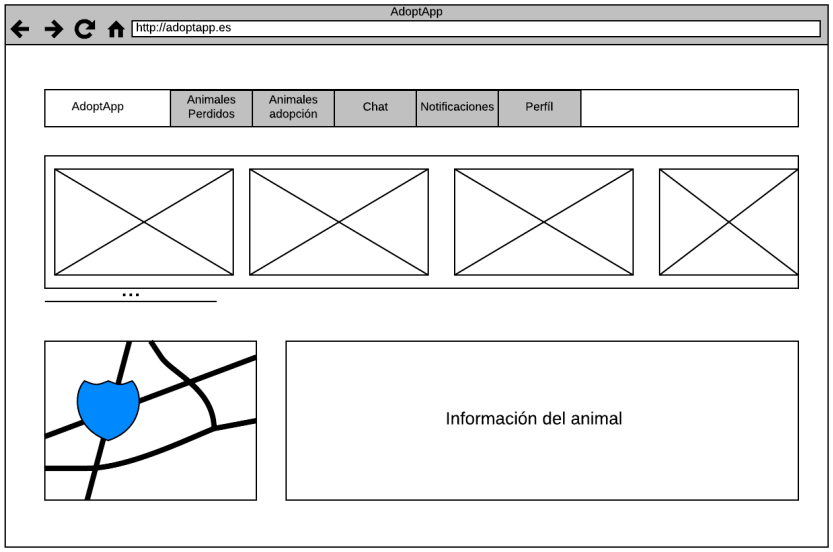


Figura 7.7: Mockup de información de un animal

Los usuarios podrán consultar la información del animal como se aprecia en la figura 7.7, mostrando de estos, sus fotografías, localización e información detallada. En la figura 7.8 puede verse como un usuario puede indicar la localización de un animal perdido en caso de que lo haya visto, pudiendo introducir un comentario.

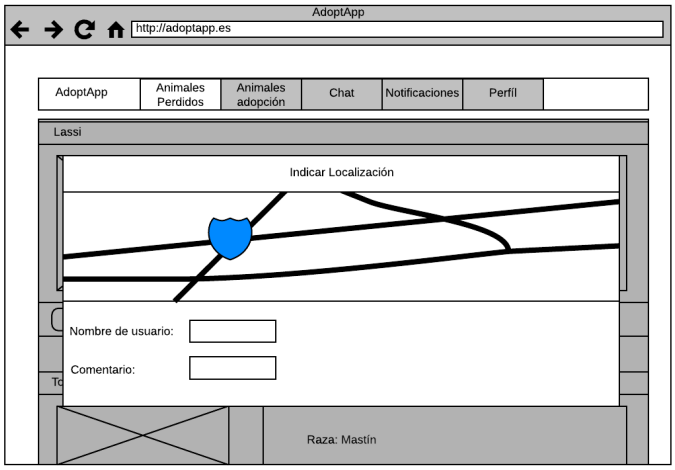


Figura 7.8: Mockup de localización de un animal perdido

Capítulo 8

Diseño de la aplicación

8.1 Introducción y Objetivos

Cuando se diseña un sistema software se determinan los componentes que lo conformarán, determinando la función de cada uno de estos así como su estructura. Se establecen además las posibles interacciones que se producen entre estos además del medio a través del que se llevan a cabo.
Durante el desarrollo de un proyecto software es inevitable la inclusión de cambios inesperados. Además, es necesario en muchas ocasiones incluir funciones en el sistema que no estaban previamente planeadas . Las modificaciones que todo esto supone implican un coste. Un buen diseño pretende reducir estos costes aumentando la flexibilidad y robustez del sistema.
En el software desarrollado se pretenden aplicar buenas prácticas de diseño que permitan reducir el tiempo de mantenimiento en el futuro. Entre los objetivos concretos que se pretenden conseguir, podemos destacar:

- Software poco acoplado, reduciendo dependencias entre componentes.
- División de funciones de forma que aumente la facilidad de comprensión del sistema.
- Reducción de la complejidad.
- Desarrollo de software reusable.

8.2 Resumen de patrones usados

Para el cumplimiento de los objetivos previamente destacados se han aplicado ciertos principios y patrones que se detallan a continuación:

Diseño por capas: Permite reducir las dependencias en el código, dividiendo el sistema en capas o niveles independientes entre sí pero que se comunican a través de unas interfaces.

La modificación de una de estas capas no altera el comportamiento de las otras. Cada una de estas capas tiene una función diferente y se dividen en:

- **Capa de acceso a datos-modelo-:** abstrae el acceso a la información, que se realiza a través de una serie de DAOs (*Data Access Objects*).
- **Capa de lógica de negocio-modelo-:** Está formada por una serie de servicios que implementan la lógica de negocio. Cada uno de estos agrupa una serie de funcionalidades relacionadas entre sí. De esta forma se mejora la comprensión del sistema y se facilita la incorporación de cambios.
- **Capa servicios:** Se divide en una serie de controladores que agrupan funcionalidades comunes. Estos utilizan DTOs (*Data Transfer Objects*) para el intercambio de información, facilitando la modificación de la información enviada y recibida en cada petición.

Uso de abstracciones: La inclusión de unos componentes en la implementación de otros se produce mediante el uso de abstracciones, evitando referirse a implementaciones concretas. Esto es posible gracias a la inyección de dependencias que permite la instanciación de componentes a partir de la interfaz que implementan. Este método reduce el nivel de acoplamiento entre las diferentes partes del sistema.

Adicionalmente, con el objetivo de facilitar la comprensión y el mantenimiento del sistema, se han aplicado los siguientes principios:

DRY (*Don‘t Repeat Yourself*) : Cada parte del sistema tiene una función única y clara. La aplicación de este principio provoca que cambios en una parte concreta de sistema no produzcan modificaciones en otras sin relación funcional.

KISS (*Keep It Simple, Stupid!*) : Reducir la complejidad del sistema para mejorar su funcionamiento y facilitar la introducción de cambios.

SOLID: Conjunto de principios que permiten un desarrollo fácil de extender.

YAGNI (*You Aren´t Gona Need It*): Desarrollar funcionalidades que se vayan a usar en un futuro próximo, evitando el desarrollo de características que no sean imprescindibles. Este principio permite reducir el tiempo de desarrollo y evita la existencia de código innecesario que aumenta la complejidad del sistema.

8.3 Arquitectura General

El sistema desarrollado consta de dos partes bien diferenciadas como se puede ver en la figura 8.1.
Por una parte cuenta con una aplicación servidor o *Backend* que se divide en tres capas. La capa de servicio proporciona una API REST que recibe las peticiones de la aplicación cliente. Esta capa redirige las peticiones a la capa de lógica de negocio que implementa el comportamiento de cada una de las funcionalidades. Esta capa interactúa con la capa de acceso a datos que se encarga de obtener y modificar la información de la Base de Datos. La función general del *Backend* es, por tanto, recibir las peticiones de la aplicación cliente y devolver una respuesta tras aplicar una lógica y consultar la BD si es necesario.
Por otra parte, el sistema incluye una aplicación cliente que contiene la interfaz con la que interactúa el usuario final. Este subsistema cuenta además con una capa de acceso al servicio encargada de enviar las peticiones necesarias a la API Rest del servidor.

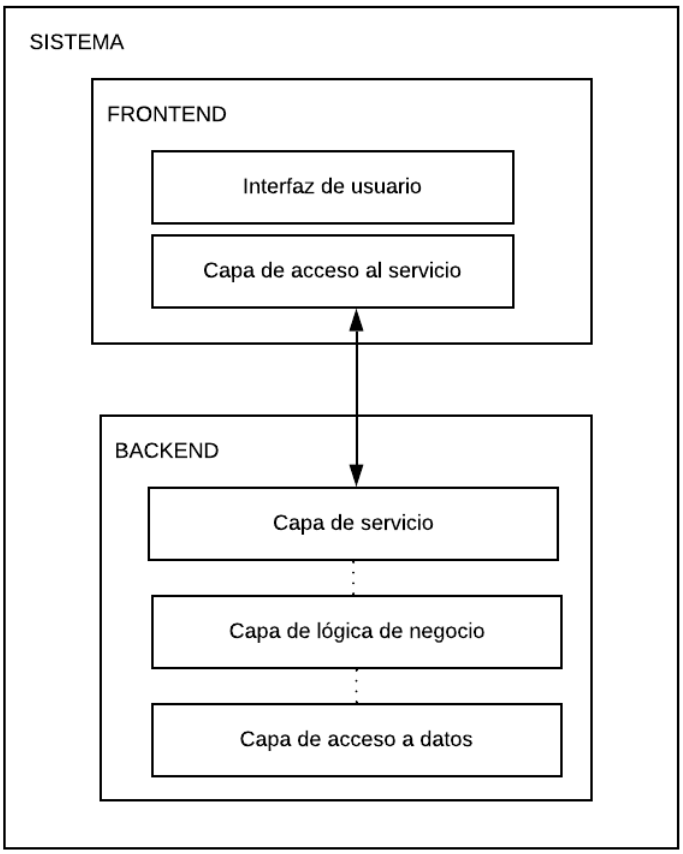


Figura 8.1: Diagrama de arquitectura general del sistema

8.4 Subsistema Servicio Web (Backend)

8.4.1 Objetivos

Este subsistema define la aplicación del lado servidor. Este ha sido desarrollado utilizando *SpringBoot*[10] y contiene la mayor parte de la lógica del sistema. El principal objetivo en el desarrollo de esta parte es, diseñar un sistema dividido en capas independientes reduciendo la dependencia de implementaciones concretas. Con esto se pretende facilitar el mantenimiento de la aplicación y reducir el tiempo de desarrollo, haciendo una división clara y comprensible de las diferentes funcionalidades.

8.4.2 Arquitectura

La arquitectura de este subsistema se puede ver en la figura 8.2

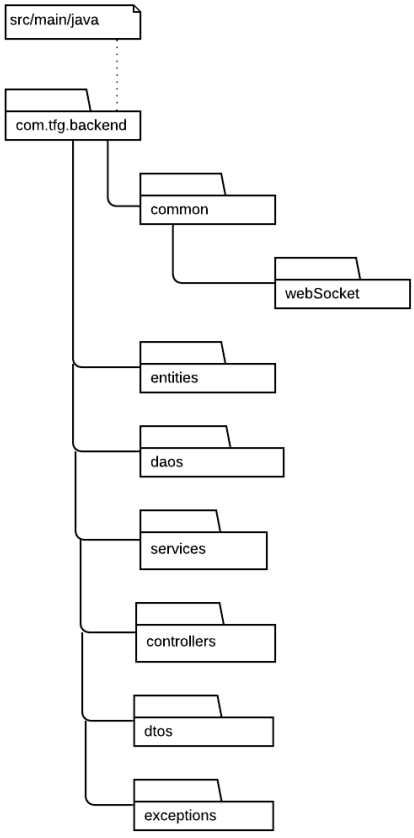


Figura 8.2: Jerarquía de ficheros del Backend

Como se puede apreciar, el sistema se divide en una serie de paquetes que agrupan elementos de diferentes capas. Los paquetes ”Entities” y ”Daos” conforman la capa de acceso

a datos. El paquete ”service” esta formado por los diferentes servicios que mantienen la lógica del negocio y en ”Controller” se encuentran los elementos que forman el API REST. A continuación se detalla, en general, cada una de estas capas y su objetivo dentro del sistema.

En primer lugar se ha desarrollado la capa de acceso a datos. Para esto, ha sido necesario en primera instancia crear las entidades que utilizará la aplicación. Estas serán mapeadas a la Base de Datos de forma automática gracias a *Spring*[41]. Para la creación, modificación y borrado de las entidades de la BD ha sido necesario la creación de la otra parte que conforma la capa de acceso a datos, esto es, los DAOs (*Data Access Objects*). Por defecto *Spring* proporciona una implementación genérica para estos. Sin embargo, se han realizado implementaciones propias con el objetivo de realizar consultas más complejas.

Sobre la capa de acceso a datos, se encuentra la capa de lógica de negocio. Esta capa esta formada por una serie de servicios divididos por ámbito de funcionalidad y son los encargados de ejecutar la lógica del sistema realizando peticiones a la capa de acceso a datos.

En última instancia, la capa servicio proporciona una API REST formada por una serie de controladores divididos, al igual que los servicios del modelo, por el tipo de funcionalidad. Esta capa recibe las peticiones de los clientes y ejecuta las funciones de los servicios necesarias.

8.4.3 Modelo del dominio

Diagrama de entidades

En la figura 8.3 se pueden ver las diferentes entidades modeladas en la capa de acceso a datos de la aplicación del lado servidor. Existen dos tipos fundamentales de entidades: clases y enumerados. Estos últimos agrupan valores que representan características de las clases. Todas las entidades cuentan con identificadores autogenerados mediante una secuencia independiente. Cada entidad se detalla a continuación:

- **Animal :** Mantiene información común entre los animales perdidos y los animales en adopción. Existen los enumerados *Tamaño*, *Color* y *AnimalGenre* para representar sus características.
- **Breed :** Mantiene las razas de los diferentes tipos de animal.
- **AdoptionAnimal :** Almacena información concreta de animales en adopción, como puede ser el tiempo que lleva en estado de adopción o su fecha de nacimiento. Utiliza el enumerado *Estado de adopción.*
- **LostAnimal :** Mantiene información sobre animales perdidos. Existe un enumerado para indicar si el animal sigue perdido o se ha encontrado.
- **AnimalPicture :** Representa imágenes de animales y mantiene información sobre estas.

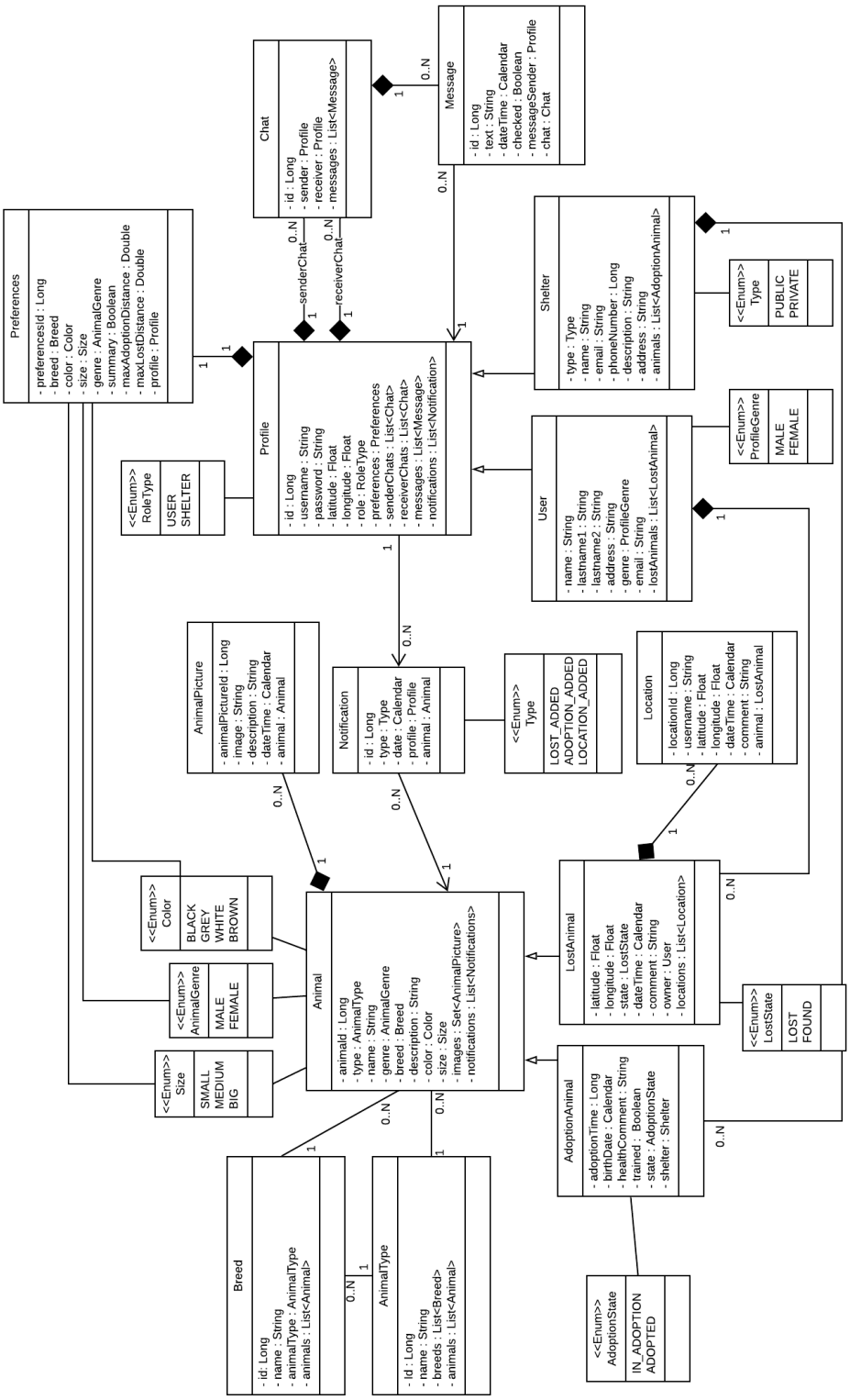


Figura 8.3: Diagrama de entidades

- **Notification :** Mantiene las notificaciones que reciben los usuarios cuando se añade un animal en adopción o se pierde un animal en su zona.

- **Location :** Representa localizaciones que los usuarios añaden sobre un animal perdido.

- **Profile :** Mantiene información de usuarios registrados. El enumerado *Tipo de rol* permite indicar si el perfil es de usuario individual o de asociación.

- **User :** Almacena información de usuarios individuales . Existe un enumerado para indicar el género del usuario.

- **Shelter :** Mantiene información de usuarios asociación. Existe el enumerado *Tipo* para indicar si la asociación es pública o privada.

- **Preferences :** Representa las preferencias de los distintos usuarios registrados. Estas incluyen la distancia máxima en la que se muestras animales en los distintos mapas así como características que el usuario busca de una animal en adopción.

- **Chat :** Almacena información de cada chat que se establece entre dos usuarios registrados.

- **Message :** Mantiene los mensajes enviados por los usuarios registrados en un chat.

Modelo de datos

En la figura 8.4 se puede ver el modelo de datos de la aplicación, este representa el conjunto de tablas que mantienen información en la base de datos. Cada una de las tablas equivale a una entidad de la capa de acceso a datos. La notación *Entity* en las diferentes entidades permite a *Spring* crear las tablas asociadas de forma automática. Estas mantienen información referente a cada entidad, incluyendo en cada caso un identificador único. Este identificador es generado de forma automática. Para ello cada entidad cuenta con un generador de secuencia propio.

Spring construye las relaciones entre entidades partiendo de ciertas notaciones: *@OneToOne @OneToMany* y *@ManyToOne*. El uso concreto de cada una de estas notaciones depende de la multiplicidad de cada relación.

Al instanciar una entidad que participa de una relación 1:N, se recupera información de las instancias con las que se relaciona dicha entidad. Esto se produce porque la política por defecto es *EAGER*. Esto puede suponer un problema de eficiencia al recuperar información de forma innecesaria. Para evitar este problema se utiliza la propiedad *fetchType = Lazy* de *@OneToMany*, lo que provoca que no se instancien entidades relacionadas en estos casos.

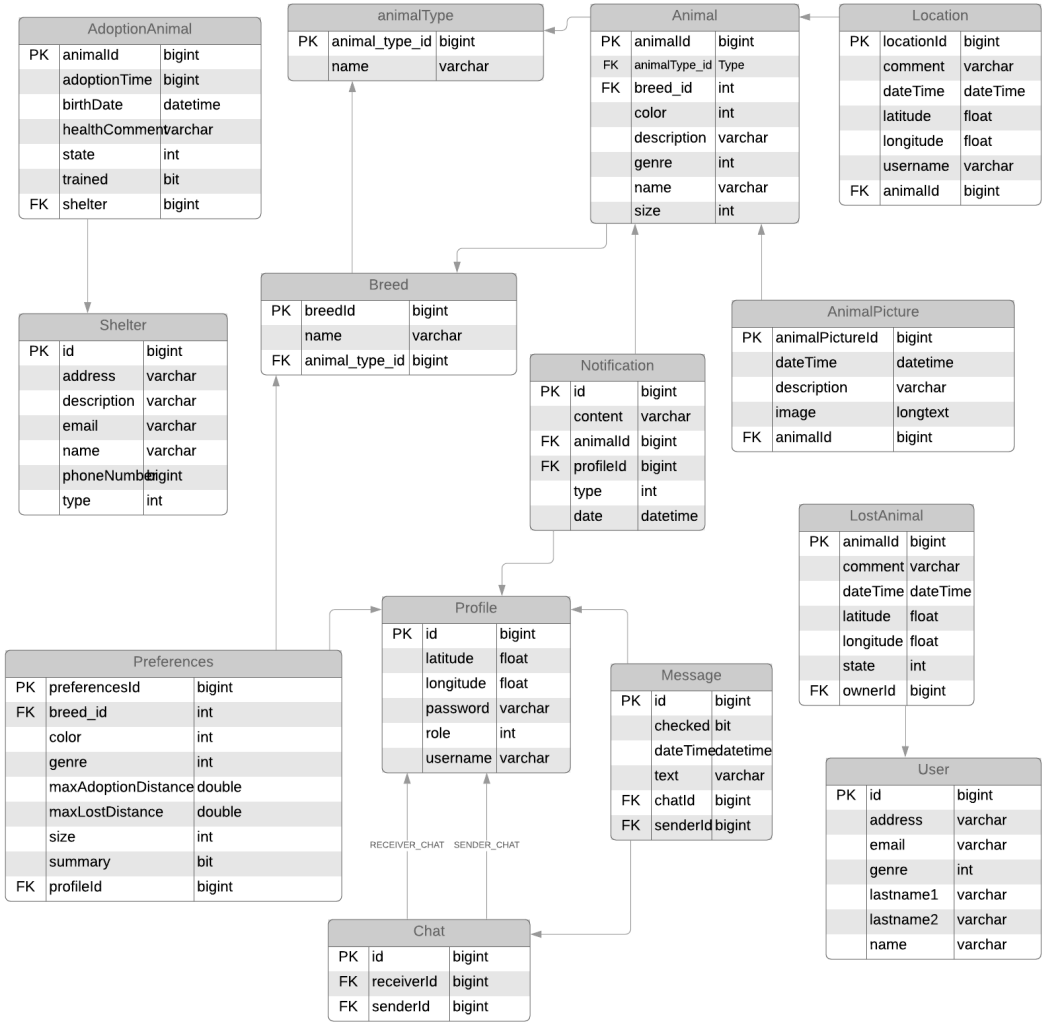


Figura 8.4: Modelo de datos

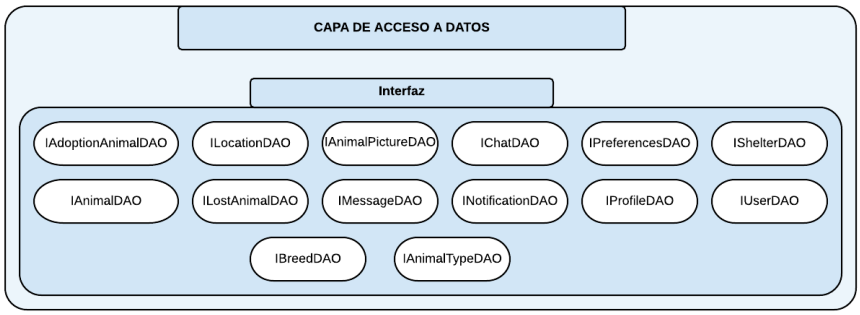


Figura 8.5: Diagrama de capa de acceso a datos

8.4.4 Capa de acceso a datos

Como se ha comentado en apartados anteriores, la capa de acceso a datos, además de estar formada por entidades, incluye una serie de elementos conocidos como DAOs (*Data Access Objects*) que se emplean para acceder a información almacenada en la Base de Datos. En la figura 8.5 se muestran las interfaces de los DAOs que se utilizan en la aplicación. Estas extienden a *PagingAndSortingRepository*, interfaz proporcionada por *Spring* que permite la ejecución de consultas básicas sobre la base de datos. Para el desarrollo de otras más complejas se ha utilizado la notación @Query que proporciona Spring y que permite establecer una consulta por defecto a cada método de la interfaz de un DAO.

PagingAndSortingRepository recibe dos parámetros: la clase de Entidad a la que accederá y el tipo del id de esta. Proporciona entonces, métodos como *findById* que busca una instancia por su identificador o *findAll* que recupera todas las instancias de una entidad concreta. Es posible además declarar métodos que busquen por características concretas de una entidad, sin necesidad de declarar una consulta de forma explícita. En la aplicación desarrollada, los métodos incluidos de esta forma son:

En **ILocationDAO:**

- List<Location> findByAnimal (Animal animal): Recupera las localizaciones indicadas sobre un animal perdido concreto.

En **IBreedDAO:**

- List<Breed> findByAnimalAnimalType (AnimalType animal): Recupera las razas de un tipo de animal concreto
- Breed findByName (String name): Busca una raza por nombre.

En **IAnimalDAO:**

- AnimalType findByName (String name): Busca un tipo de animal por nombre.

En **INotificationDAO:**

- List<Notification> findByProfileId (Long ProfileId): Recupera las notificaciones de un determinado usuario.

En **IPreferencesDAO:**

- List<Preferences> findBySummary(Boolean summary): Recupera las preferencias de usuario en función de sí quiere o no recibir boletines semanales.

En **IProfileDAO:**

- Profile findByUsername(String username): Recupera la información de un perfil a partir del nombre de usuario

En **IUserDAO:**

- User findByUsername(String username): Recupera la información de un usuario individual a partir del nombre de usuario.

La interfaz PagingAndSortingRepository permite además, devolver el resultado de las consultas de forma paginada y/o ordenada.

En muchos casos es necesaria la creación de consultas de mayor complejidad. La notación *@Query* permite declarar una consulta por defecto en cada función de la interfaz, pudiendo ser implementada en un futuro para proporcionar consultas alternativas. Estas son declaradas en lenguaje *JPQL*[6]. A continuación se detallan las consultas de este tipo añadidas a los *DAOs* y su función:

En **IAnimalDAO**:

- searchAnimalsByFilter: Devuelve la lista de animales que satisfagan las características indicadas en el filtro.

En **IChatDAO**:

- findChatByUsers: Devuelve el chat entre dos usuarios concretos.
- getUserChats: Devuelve los chats en los que participa un usuario.

En **ILostAnimalDAO**:

- searchLostAnimalsByDistance: Devuelve los animales perdidos á una distancia máxima del usuario. El resultado se presenta paginado.
- searchLostAnimalsInArea: Devuelve los animales perdidos á una distancia máxima del usuario.
- searchLostAnimalsDistances: Devuelve las distancias a las que se encuentra un usuario de los animales perdidos cercanos
- countLostAnimals: Devuelve el numero de animales perdidos cercanos a un usuario concreto

En **IMessageDAO**:

- getMessagesFromChat: Devuelve el listado de mensajes en un chat concreto.

- getLastMessageFromChat: Devuelve el ultimo mensaje enviado en un chat concreto.

En **IProfileDAO**:

- findByLostPreferences: Devuelve el listado de usuarios cercanos a un animal perdido.

En **IShelterDAO**:

- getSheltersInArea: Devuelve el listado de asociaciones cercanas a un usuario concreto.
- getDistances: Devuelve las distancias entre un usuario concreto y las asociaciones cercanas

En **IUserDAO**:

- findByAnimalPreferences: Devuelve el listado de usuarios que cuentan con unas preferencias de animal concretas.

Puesto que la aplicación requiere del cálculo de distancias entre coordenadas se ha utilizado la función *STDistanceSphere* en alguna de las consultas. Esta permite obtener la distancia entre dos puntos geográficos.

8.4.5 Capa Lógica de Negocio (Modelo)

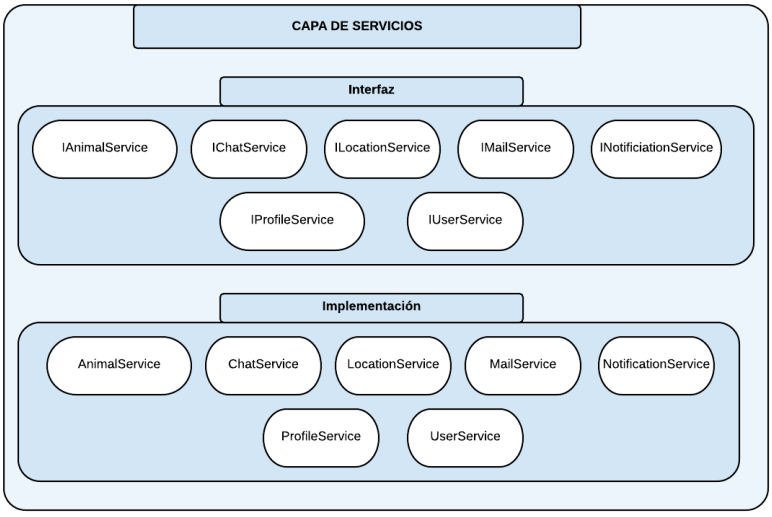


Figura 8.6: Diagrama de capa de lógica de negocio

Esta capa está formada por un conjunto de servicios, divididos por el tipo de funcionalidad que implementan. Se divide, más concretamente en interfaces de servicio y sus implementaciones concretas. La estructura de la capa se puede ver en la figura 8.6. Las interfaces declaran

las funciones concretas que lleva a cabo cada servicio. Las clases de implementación aportan la lógica concreta de cada una de estas funcionalidades.

Los servicios reciben peticiones de la capa de servicio y ejecutan una serie de operaciones en consecuencia. Entre estas se encuentran la consulta y/o modificación de información en la base de datos. Para ello utilizan las funciones que aportan los DAOs.

Spring inyecta la implementación concreta de estos servicios en la capa de servicio gracias a la notación *@Service*. Esta se incluye en cada implementación concreta y permite que sea instanciada conociendo únicamente la interfaz a la que implementa. Esto supone un diseño más modular y menos acoplado.

A continuación se detalla la función principal de cada servicio y las operaciones que lleva a cabo:

AnimalService

Funciones de consulta y modificación de información asociada a los animales.

- List<AnimalMarkerDTO> **getNearbyAdoptionAnimals**(ProfileDTO profile) : Devuelve las localizaciones de animales en adopción cercanos.
- List<Animal> **searchAdoptionAnimalByFilter**(AdoptionAnimalFilterDTO filter) : Devuelve los animales en adopción que satisfagan las características indicadas en el filtro.
- EnumsDTO **getEnumValues**() : Devuelve los valores de los enumerados que representan caraterísticas de un animal.
- LostAnimalsPageDTO **getAllLostAnimals**(int page) : Devuelve los animales perdidos paginados.
- LostAnimalsInAreaDTO **getAnimalInArea**(String token) : Devuelve los animales perdidos en la zona de un usuario.
- AnimalDTO **getLostAnimalInfo**(Long animalId) : Devuelve la información de un animal perdido
- AdoptionAnimalsPageDTO **getAdoptionAnimals**(SearchAdoptionAnimalsDTO searchAdoptionAnimalsDTO) : Devuelve el listado de animales en adopción paginado.

ChatService

Funciones que modifican y consultan información asociada a Chats.

- List<Message> **storeMessage** (MessageDTO messageDTO): Almacena un mensaje en la Base de datos.

- List<Message> **getMessages** (ChatDTO chatDTO): Obtiene los mensajes de un chat.
- ChatDTO **startChat**(ChatDTO chatDTO) : Crea un chat entre dos usuarios registrados.
- List<ProfileChatReducedDTO> **getChats**(String userToken): Obtiene los chats activos de un usuario .

LocationService

Operaciones sobre localizaciones de animales perdidos.

- ReturnedLocationDTO **addLocation**(LocationDTO locationDTO): Añade una localización á un animal perdido.

MailService

Operaciones que gestionan el envío de correos electrónicos.

- void **sendEmail**(List<AdoptionAnimal> animals, User user): Envía un correo electrónico á un usuario.
- void **sendSummaries**(): Envía boletines semanales por correo electrónico a los usuarios por correo electrónico.

NotificationService

Operaciones sobre las notificaciones que reciben usuarios registrados.

- List<Notification> **getNotifications**(String userToken): Obtiene las notificaciones de un usuario concreto.
- List<Notification> **deleteNotification**(Long notificationId, String userToken) : Borra una notificación de un usuario .

ProfileService

Operaciones sobre los perfiles de usuario. Comunes entre los diferentes tipos de usuario.

- ProfileDTO **login**(String username, String password): Autentica a un usuario con sus credenciales.
- Profile **findById**(Long userId): Busca a un usuario autenticado por su id.
- ProfileDTO **setLocation**(LocationDTO locationDTO): Añade la localización á un perfil.

- ProfileDTO **registerProfile**(ProfileDTO profileDTO): Registra un perfil en el sistema.
- ProfileDTO **updateProfile**(ProfileDTO profileDTO): Actualiza la información de un perfil.
- ProfileEnumsDTO **getProfileFromToken**():Obtiene un perfil de usuario a partir de un token de autenticación.
- ProfileEnumsDTO **getProfileEnums**(): Obtiene el valor de los enumerados que representan las caracteristicas de un perfil.

UserService

Operaciones llevadas a cabo por un usuario individual

- AnimalDTO **addLostAnimal** (AnimalDTO animalDTO): Crea un animal perdido en el sistema.
- User **getUserFromToken**(String userToken): Obtiene un perfil de tipo usuario a partir del token de autenticación.
- LostAnimalPageDTO **getUserLostAnimals**(String userToken, int page): Obtiene los animales perdidos de un usuario concreto.
- LostAnimalPageDTO **deleteLostAnimal**(DeleteAnimalDTO deleteAnimalDTO): Borra un animal perdido del sistema.
- UserPreferencesDTO **getPreferences**(String token): Obtiene la preferencias de un usuario.
- UserPreferencesDTO **editPreferences**(UserPreferencesDTO UserPreferencesDTO): Modifica las preferencias de un usuario .

ShelterService

Operaciones llevadas a cabo por usuarios de tipo asociación.

- ReturnedAdoptionAnimalDTO **addAnimal**(AnimalDTO animalDTO): Añade un animal en adopción.
- AdoptionAnimalsPageDTO **getShelterAdoptionAnimals**(SearchShelterAnimalsDTO param): Obtiene los animales en adopción en una asociación concreta.
- AdoptionAnimalsPageDTO **deleteAnimal**(DeleteAnimalDTO deleteAnimalDTO): Borra un animal en adopción.

- ReturnedAdoptionAnimalDTO **editAnimal**(AnimalDTO animalDTO) : Modifica la información de un animal en adopción.
- Shelter **getShelterFromToken**(String userToken): Obtiene un perfil de tipo asociación a partir de un token de autenticación.
- ShelterPreferencesDTO **editPreferences**(ShelterPreferencesDTO shelterPreferencesDTO): Modifica las preferencias de una asociación.
- ShelterPreferencesDTO **getPreferences**(String token): Obtiene las preferencias de una asociación.

A continuación se describen una serie de Excepciones que se pueden producir en las funciones anteriores ante determinados casos de error:

- **InstanceNotFoundException :** Se produce cuando se intenta recuperar una instancia concreta de una Entidad que no existe en la Base de Datos.
- **ForbiddenException :** Se genera al intentar ejecutar una función no permitida por el usuario autenticado.
- **DuplicatedUserException :** Se produce al intentar registrar un perfil con un nombre de usuario ya existente en la Base de Datos.
- **IncorrectLoginException :** Se produce al intentar autenticarse con un nombre de usuario o contraseña incorrectos.

  Estas excepciones serán capturadas por la capa de servicio y transformadas en códigos de respuesta HTTP, para ser enviados posteriormente al *Frontend*.

8.4.6 Capa de Servicio

Esta capa tiene como objetivo fundamental recibir peticiones de la aplicación cliente y ejecutar funciones de los servicios en consecuencia. Para realizar esta tarea, la capa de servicio ofrece una serie de *endpoints*. Cada uno de estos cuenta con una función asociada que ejecutará la lógica correspondiente a la petición. Para identificar dichos endPoints y la función que ejecutan, *Spring*[41] proporciona las notaciones *@PostMapping*, *GetMapping* y *@PutMapping*. Estas representan a los tipos de petición siguientes:

- **POST :** Creación o Borrado de información
- **PUT :** Modificación de información
- **GET :** Consulta de información

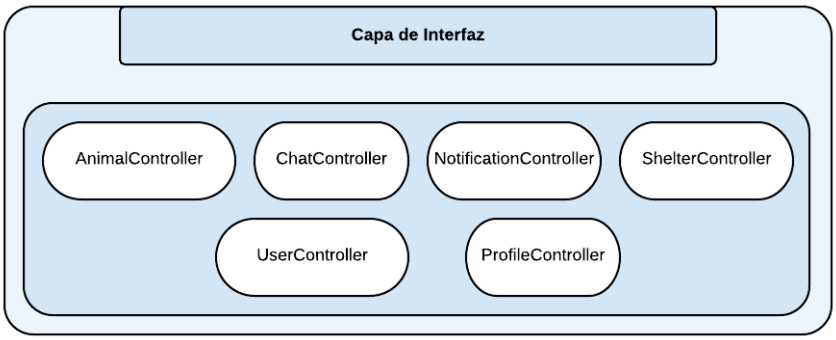


Figura 8.7: Diagrama de capa de servicio

Ante una petición concreta, el controlador correspondiente ejecuta una o varias funciones de los servicios. Para ello Spring inyecta implementaciones concretas de los servicios en los controladores mediante la notación *@Autowired* sobre la declaración de las interfaces que implementan los servicios. Esto evita tener que instanciar implementaciones concretas reduciendo el acoplamiento del código.

Los *endpoints* y las peticiones que reciben se detallan a continuación:

Animal Controller :

Recurso	**Método**	**Excepciones**
/animal/getAll	GET	Ninguna
/animal/getNearbyAdoptionAnimals	GET	InstanceNotFoundException
/animal/filterAdoptionAnimals	GET	Ninguna
/animal/getInfo	GET	InstanceNotFoundException
/animal/getTypes	GET	Ninguna
/animal/getLostAnimals	GET	Ninguna
/animal/addLocation	POST	InstanceNotFoundException
/animal/locationsPage	GET	IntanceNotFoundException
/animal/locations	GET	InstanceNotFoundExcption
/animal/searchByDistance	GET	InstanceNotFoundException
/animal/lostAnimalsInArea	GET	InstanceNotFoundException
/animal/lostAnimalInfo	GET	Ninguna

- **/animal/getAll:** Obtiene el listado completo de animales en adopción.
- **/animal/getNearbyAdoptionAnimals:** Obtiene la posición de los animales en adopción cercanos al usuario.

- **/animal/filterAdoptionAnimals:** Devuelve el listado de animales en adopción que satisfagan las características indicadas en un filtro .
- **/animal/getTypes:** Devuelve el valor de los enumerados que representan características de los animales. Estos son: Generos, Tamaños, Razas, Colores, Estados de animales en adopción y Estados de animales perdido.
- **/animal/getLostAnimals:** Devuelve un listado paginado de animales en adopción.
- **/animal/addLocation:** Añade una localización a un animal perdido.
- **/animal/locationsPage:** Recupera las localizaciones paginas indicadas sobre un animal perdidos.
- **/animal/locations:** Obtiene el listado completo de localizaciones indicadas sobre un animal perdido.
- **/animal/searchByDistance:** Devuelve un listado paginado de animales perdidos cerca de un usuario.
- **/animal/lostAnimalsInArea:** Obtiene el listado completo de animales perdidos cerca de un usuario.
- **/animal/lostAnimalInfo:** Recupera la información de un animal perdido.

Profile Controller :

Recurso	**Método**	**Excepciones**
/profile/register	POST	DuplicatedUserException
/profile/login	GET	IncorrectLoginException
/profile/update	PUT	ForbiddenException, InstanceNotFoundException
/profile/setLocation	PUT	ForbiddenException, InstanceNotFoundException
/profile/getInfo	GET	InstanceNotFoundException
/profile/enums	GET	Ninguna

- **/profile/register:** Registra un perfil de usuario individual o de asociación.
- **/profile/login :** Inicia la sesión de un usuario.
- **/profile/update:** Actualiza la información de perfil de un usuario individual o asociación

- **/profile/setLocation:** Añade una localización a un perfil de usuario individual o asociación
- **/profile/getInfo:** Devuelve la información de perfil de un usuario individual o asociación
- **/profile/enums:** Devuelve el valor de los enumerados que representan características de los diferentes perfiles. Estos son: Genero y Tipo de Asociación.

Shelter Controller :

Recurso	Método	Excepciones
/shelter/list	GET	InstanceNotFoundException, ForbiddenException
/shelter/animal/add	POST	InstanceNotFoundException, ForbiddenException
/shelter/animal/edit	PUT	InstanceNotFoundException, ForbiddenException
/shelter/animal/delete/	POST	InstanceNotFoundException, ForbiddenException
/shelter/shelterList	GET	InstanceNotFoundException
/shelter/shelterDistance	GET	InstanceNotFoundException
/shelter/preferences	GET	InstanceNotFoundException, ForbiddenException
/shelter/preferences	PUT	InstanceNotFoundException, ForbiddenException

- **/shelter/list:** Devuelve el listado paginado de animales en la asociación.
- **/shelter/animal/add:** Añade un animal en adopción en la asociación.
- **/shelter/animal/edit:** Modifica la información de un animal en adopción.
- **/shelter/animal/delete:** Borra la información de un animal en adopción.
- **/shelter/shelterList:** Devuelve el listado completo de asociaciones.
- **/shelter/shelterDistance:** Devuelve la posición de las asociaciones cercanas al usuario.
- **/shelter/preferences (GET):** Devuelve las preferencias de una asociación.
- **/shelter/preferences (PUT):** Modifica las preferencias de una asociación

User Controller :

Recurso	Método	Excepciones
/user/addLostAnimal	POST	InstanceNotFoundException, ForbiddenException
/user/getAnimals	GET	InstanceNotFoundException, ForbiddenException
/user/deleteLost	POST	InstanceNotFoundException, ForbiddenException
/user/preferences	GET	InstanceNotFoundException, ForbiddenException
/user/preferences	PUT	InstanceNotFoundException, ForbiddenException

- **/user/addLostAnimal:** Añade un animal perdido.
- **/user/getAnimals:** Obtiene el listado de animales perdidos de un usuario
- **/user/deleteLost:** Borra la información de un animal perdido
- **/user/preferences (GET):** Obtiene las preferencias de un usuario
- **/user/preferences (PUT):** Modifica las preferencias de un usuario

Chat Controller :

Recurso	Método	Excepciones
/chat/history	GET	InstanceNotFoundException, ForbiddenException
/chat/start	POST	InstanceNotFoundException
/chat/active	GET	InstanceNotFoundException, ForbiddenException

- **/chat/history:** Devuelve el listado de mensajes de un chat concreto.
- **/chat/start:** Inicia un chat entre dos usuarios registrados.
- **/chat/active:** Devuelve el listado de chats activos de un usuario.

Notification Controller :

Recurso	Método	Excepciones
/notification/get	GET	InstanceNotFoundException, ForbiddenException
/notification/delete	POST	InstanceNotFoundException, ForbiddenException

- **/notification/get:** Obtiene el listado de notificaciones de un usuario.
- **/notification/delete:** Borra una notificación.

Cada una de las excepciones indicadas previamente es capturada por un manejador definido en los controladores. *Spring*[10] proporciona la notación *ExceptionHandler* para indicarlos. Estos capturan Excepciones concretas y devuelve un código de estado *HTTP* y un mensaje de error que sera enviados como respuesta a la aplicación cliente.

A continuación se muestran los mensajes de error HTTP asociados a cada excepción:

- ForbiddenException : **403 Forbbiden**
- InstanceNotFoundException : **404 Not Found**
- DuplicatedUserException : **400 Bad Request**
- IncorrectLoginException : **404 Not Found**

8.4.7 Integración con Servicio de Correo

Para el envío de correos electrónicos desde el *Backend* se utiliza *JavaMail*. Esta librería proporciona una serie de clases y funciones que permiten establecer una conexión con un servidor de correo y enviar peticiones. Se ha creado el servicio *MailService* en la capa de lógica de negocio para gestionar el envío de correos. Las clases de *JavaMail* involucradas en el proceso son las siguientes:

- **Properties:** Permite indicar las propiedades de la conexión con el servidor de correo.
- **Session:** Obtiene una conexión con el servidor de correo, para ello recibe un objeto de la clase *Properties* con la información del servidor además de las credenciales de acceso.
- **MimeMessage:** Contiene la información del mensaje. Permite indicar el tipo del contenido con la funcion *setContent*. En la aplicación el contenido de cada mensaje es enviado en formato *HTML*.
- **Transport:** Permite enviar el mensaje a través de la función *Transport.send(MimmeMessage Message)*

El servicio incorpora estas clases y contiene dos funciones para ejecutar el envío de correos:

- **SendMail**: Gestiona el envío de un correo electrónico. Establece la conexión con el servidor, construye el mensaje y lo envía utilizando las clases indicadas previamente.

- **SendSummaries**: Obtiene el correo de los usuarios que reciben boletines y la información de cada uno de estos. Envía los correos electrónicos a cada usuario a través del método *sendMail*. Esta se ha establecido como una tarea programada para ser ejecutada semanalmente. Para ello se ha anotado con *@Scheduled*.

:

8.4.8 Websockets

Como se ha establecido en los requisitos, la aplicación cuenta con un sistema de chat que permite la comunicación entre usuarios. Para el desarrollo de esta funcionalidad se han empleado *websockets*. Esta tecnología permite crear un canal de comunicación bidireccional y *full-duplex* sobre un único *socket* TCP.
Para su implementación, se han incluido las siguientes clases en el proyecto:

- **WebSocketConfig**: Se anota con *@EnableWebSocketMessageBroker* para habilitar los *websockets* en la aplicación. En este archivo se configura y crea un agente o *Broker* en memoria encargado de enviar y recibir mensajes. Para ello se utiliza la función *ConfigureMessageBroker* que establece los destinos a los que se envían y reciben peticiones y determina el *endpoint* gestionado por el controlador.

- **CustomPrincipal**: Implementa la interfaz *Principal* y define su estructura. En la aplicación los objetos de esta clase encapsulan el nombre del usuario al que se destina un mensaje determinado.

- **CustomHandShakeHandler** : Intercepta los mensajes antes de enviarlos al controlador. En esta clase se crea un objeto de tipo *Principal* a partir del contenido de la petición *HTTP* recibida y se envía al controlador como parámetro.

Por otra parte se ha creado el controlador *ChatController* encargado de gestionar la recepción de mensajes. Este cuenta con un método anotado con *@MessageMapping* que indica el prefijo de las peticiones que serán interceptadas por el método. Los mensajes son enviados a determinados *topics* a los que los clientes se pueden suscribir. Los *topics* disponibles y sus rutas se indican en el fichero *WebSocketConfig* nombrado anteriormente. Cuando un mensaje es enviado a un *topic*, todos los usuarios suscritos los reciben. El método del controlador cuenta también con la anotación *@SendTo* que indica la ruta del *topic* a través del cual será enviado el mensaje, p.e */topic/all*.

Es posible enviar un mensaje a un usuario concreto suscrito a un *topic* determinado. Para ello se utiliza en el controlador la función *ConvertAndSendToUser*. En la aplicación desarrollada, los mensajes son enviados de esta forma. Esta función recibe un identificador de usuario o, en el caso de la aplicación desarrollada el id de un chat y modifica la ruta a la que es enviado el

mensaje añadiendo el identificador al final de esta, p.e */topic/all/chatId*. De esta forma aunque todos los mensajes sean enviados a un mismo *topic*, únicamente los usuarios suscritos a la ruta indicada reciben el mensaje.

8.5 Subsistema Aplicación Web(Frontend)

8.5.1 Objetivos

Este subsistema representa la aplicación cliente desarrollada en *React*[12]. Proporciona la interfaz de usuario y realiza peticiones a la aplicación del lado servidor. Al desarrollar esta parte del sistema se pretenden cumplir los objetivos de diseño establecidos previamente. Se busca aplicar buenas prácticas que permitan facilitar el propio proceso de desarrollo del sistema así como su posterior mantenimiento, además de facilitar la comprensión de sus componentes. Por tratarse de una aplicación que realiza peticiones al *Backend* se pretende, por otra parte, minimizar el número de solicitudes reduciendo así, la carga de trabajo del servidor.

8.5.2 Arquitectura

La arquitectura general del sistema se puede ver en la figura 8.8. Este cuenta con dos partes principales bien diferenciadas: la capa de interfaz de usuario que contiene los diferentes componentes que conforman el apartado visual y la capa de acceso a datos que mantiene las peticiones que el subsistema realiza al *Backend*. Esta estructura facilita la comprensión del sistema y el proceso de modificación de sus componentes.
La capa de acceso al servicio esta formada por una serie de servicios que contienen las solicitudes que realiza la aplicación cliente al *Backend*. Estas construyen peticiones *HTTP* que serán enviadas a la capa de servicio de la aplicación del lado servidor. La información que es necesario enviar en peticiones de tipo *PUT* o *POST* es encapsulada en formato *JSON*. Tras enviar una petición, las funciones de la capa de acceso a datos interceptan la respuesta del servidor para que su información pueda ser utilizada por los componentes.

La capa de interfaz de usuario esta compuesta por una serie de componentes. Estos se distribuyen en módulos como se puede apreciar en la figura 8.9. Estos agrupan componentes que comparten ámbito de funcionalidad. A continuación se explica el contenido de cada uno de los módulos:

- **Animal :** Contiene los componentes que intervienen en la creación, consulta y modificación de animales. Este se divide a su vez en otros dos módulos para la gestión de animales perdidos y animales en adopción.
- **App :** Componentes que se muestran en la totalidad de la aplicación, como puede ser el menú superior.

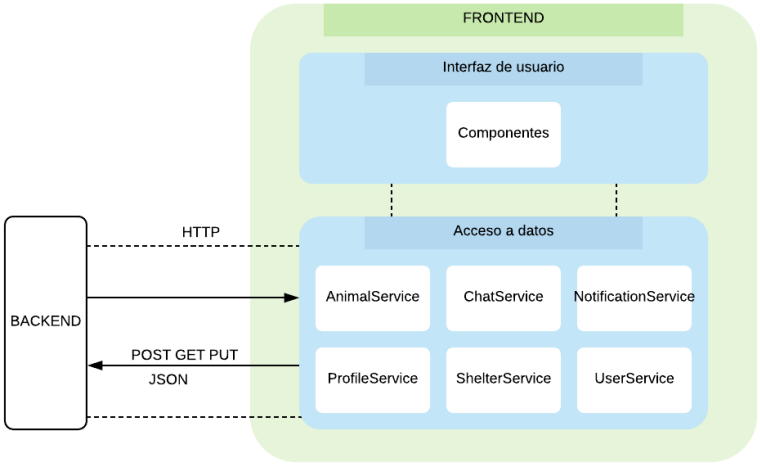


Figura 8.8: Diagrama de arquitectura del Frontend

- **Chat :** Componentes que conforman un chat o el listado de chats activos.
- **Common :** Componentes que se reutilizan en diferentes módulos.
- **MainList :** Componentes que conforman listas de animales en la aplicación.
- **Shelter :** Componentes que conforman la interfaz de aquellas funciones que puede llevar a cabo únicamente una asociación.
- **User :** Componentes que conforman la interfaz de aquellas funciones que puede llevar a cabo cualquier perfil así como únicamente un usuario individual.

Cuando un usuario accede a la aplicación se obtiene un componente principal *App*. Este contiene un elemento *Router* que permite asociar rutas o *URLs* a cada componente. El direccionamiento desde los componentes se produce, por tanto, mediante la modificación de estas rutas de acceso.

8.5.3 Capa Web

Como se ha indicado previamente, la interfaz de usuario está formada por una serie de componentes. Estos son ficheros *JavaScript* que contienen la parte visual de la aplicación y en algunos casos una parte lógica. Cada componente contiene una función *render* que muestra sus elementos visuales. Estos son declarados fundamentalmente utilizando código *HTML* al que se le aplican unos estilos *CSS*. Cuando se recupera un componente se ejecuta su método *render* y se muestra su contenido. Es posible incluir unos componentes dentro de otros, permitiendo su reutilización. En la aplicación, los componentes del módulo *Common* son reutilizados en distintos puntos de esta forma.

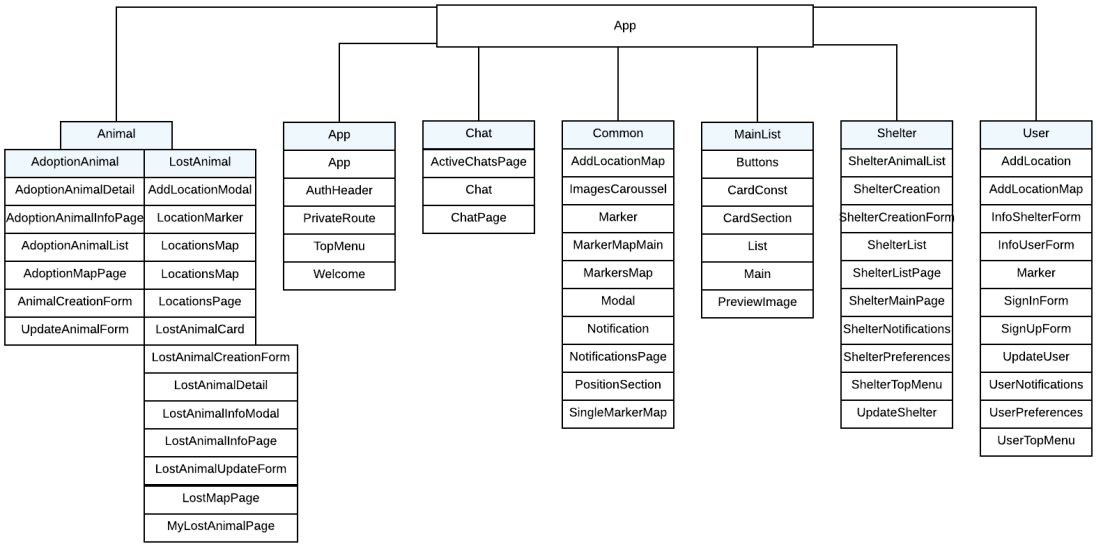


Figura 8.9: Diagrama de componentes del Frontend

En la aplicación, existen componentes que modifican y/o obtienen información de la base de datos como pueden ser *LostAnimalCreationForm* o *ShelterAnimalList*. Estos necesitan realizar peticiones al *Backend* por lo que importan funciones declaradas en la capa de de acceso a datos del *Frontend*. De esta forma, pueden ser ejecutadas por el propio componente ante ciertas acciones del usuario o como parte de su ciclo de vida.

Cada componente mantiene un estado formado por información utilizada por este. En el caso, por ejemplo, de *ShelterCreationForm* el estado lo forman los diferentes valores que se introducen en el formulario. Esta información parte de un estado inicial y se modifica con las acciones del usuario. Puede además, ser compartida entre componentes. Para ello es posible insertar el estado de un componente en las propiedades de otro. En ocasiones, es necesario compartir este estado entre varios componentes de la aplicación. La utilización de propiedades en este caso dificulta el proceso y la comprensión del código. Para solventar esta problemática se ha incluido *Redux*.

Redux permite mantener un estado global en la aplicación. Este se distribuye en una serie de componentes conocidos como reductores. Esta información es accesible por cualquier componente. Los reductores implementados son los siguientes:

- **LostAnimalReducer**: Mantiene el listado de animales perdidos y la información un animal perdido seleccionado.
- **UserReducer**: Mantiene la información del usuario autenticado para ser utilizada en diferentes puntos de la aplicación.

Para modificar el estado de *Redux* es necesario lanzar una acciones que realizan peticio-

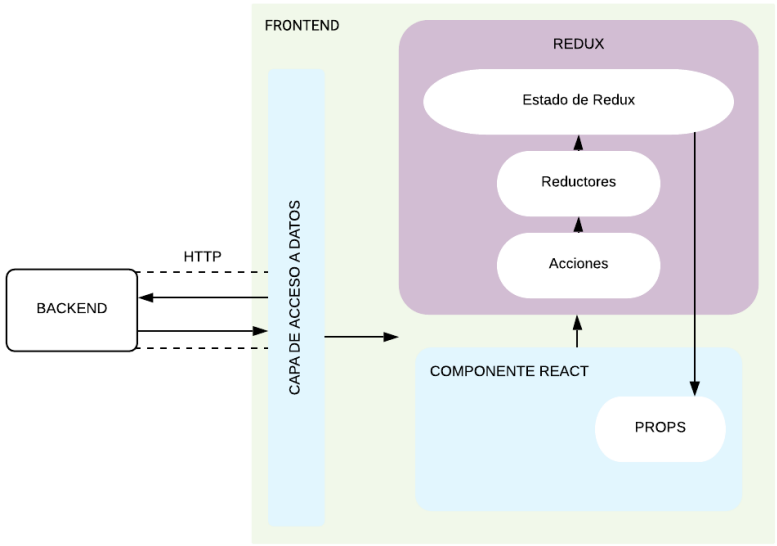


Figura 8.10: Diagrama de React con Redux

nes al *Backend* y envían la respuesta a los reductores. Estos interpretan dicha información y modifican el estado en consecuencia.

Los componentes se vinculan al estado global de la aplicación mediante una función *connect*. Esta recibe, a su vez, una función *mapStateToProps* que se encarga de inyectar la información del estado global en las propiedades del propio componente.

Además de proporcionar una forma de compartir un estado global entre componentes, *Redux* permite mantener cacheados resultados de peticiones al *Backend*, evitando tener que repetir solicitudes concretas.

Cabe destacar el uso de la librería *react-intl* para facilitar la internacionalización de la aplicación. Esta incluye un componente *<FormattedMessage>* que permite añadir textos en los componentes, obtenidos de los diferentes ficheros de traducción.

El diseño de la aplicación es *responsive* de forma que la interfaz de usuario se adapta a diferentes tipos de pantallas. Para ello en los archivos *CSS* se han incluido *media queries* que permiten indicar diferentes propiedades en función del tamaño de la pantalla. Adicionalmente se han utilizado unidades de medida dependientes del tamaño de la pantalla como son *vh* y *vw*.

8.5.4 Integración con Servicio de Mapas

La aplicación cliente cuenta con integración con el servicio de mapas de *Google*. Para ello se ha incorporado en la capa de interfaz de usuario un componente *GoogleMapReact* obtenido a través de *Npm*. Este recibe una clave de acceso al API de *Google* y como resultado renderiza

un mapa en la interfaz. Es posible incorporar otros componentes dentro del mapa de forma que se muestren en su interior. En la aplicación se ha creado el componente *Marker* en la capa de interfaz de usuario que renderiza un marcador en el mapa.

Tras pulsar en un punto del mapa se muestra el marcador en la posición seleccionada, además el componente *GoogleMapReact* realiza una petición al API de *Google* solicitando las coordenadas del punto seleccionado. Estas son almacenadas en el estado de los distintos componentes y enviadas en peticiones al *Backend* en caso de que sea necesario.

Para obtener las posiciones de los marcadores que se muestran en los distintos mapas, los componentes importan y ejecutan las funciones de la capa de acceso al servicio necesarias para solicitar las coordenadas al *Backend*. Este responde a dichas peticiones con las coordenadas de las posiciones solicitadas. Posteriormente estas son utilizadas para mostrar los marcadores en los mapas.

8.5.5 Integración con Servicio de Mensajería

Como se ha determinado en el apartado de diseño del *Backend* la aplicación cuenta con un sistema de mensajería que permite la comunicación entre usuarios. Esta funcionalidad se ha implementado mediante *websockets*. Este concepto se detalla más en profundidad en el apartado 8.4.8. Para la implementación de esta funcionalidad en el *Frontend* se ha creado en la capa de interfaz de usuario un componente *Chat*. Este está compuesto, a su vez, por los siguientes componentes:

- **SockJsClient:** Gestiona la recepción de mensajes. Para ello se indica la ruta de conexión al *broker* configurado en la aplicación del lado servidor. Esto permite que el componente establezca un canal de comunicación. Este recibe además el *topic* al que se suscribe el chat y la acción que se desencadena cuando se recibe un mensaje. Todos los clientes se suscriben al mismo *topic* y incorporan al final de la ruta correspondiente el id del chat al que acceden.

- **TalkBox:** Gestiona el envío de mensajes. Recibe una lista de mensajes y los renderiza en un chat en la interfaz. Obtiene además el nombre del usuario actual, del usuario destino y la función a ejecutar con el envío del mensaje. Esta función se encuentra en la capa de acceso a servicios del *Frontend* y envía el mensaje al *endpoint* configurado en el *Backend*.

Capítulo 9

Implementación

9.1 Software Requerido

En esta sección se detalla el software necesario para llevar a cabo el proyecto:

Para la construcción del *Backend :*

- El paquete para desarrolladores **Java SE Runtime Environment 8+**.
- El plugin **SpringTools 4.3.1+** de eclipse para la generación del proyecto *Springboot*.
- La base de datos **h2 1.4.2+**.
- Un sistema de base de datos **Mysql 8+**.
- La herramienta para construcción de proyectos **Maven 3.5.3+**.

Para el desarrollo del *Frontend :*

- El administrador de paquetes **npm 6.9+**.
- Un entorno de ejecución **node.js 10.16+**.

9.2 Estructura

La estructura de ficheros del proyecto se divide en dos partes bien diferenciadas. Por una parte nos encontramos el directorio *AdoptApp* que contiene los ficheros correspondientes al *Backend*. Los ficheros de configuración y construcción del *Frontend* se encuentran el el directorio *AdoptApp-Frontend*. A continuación se describe cada uno de estos directorios.

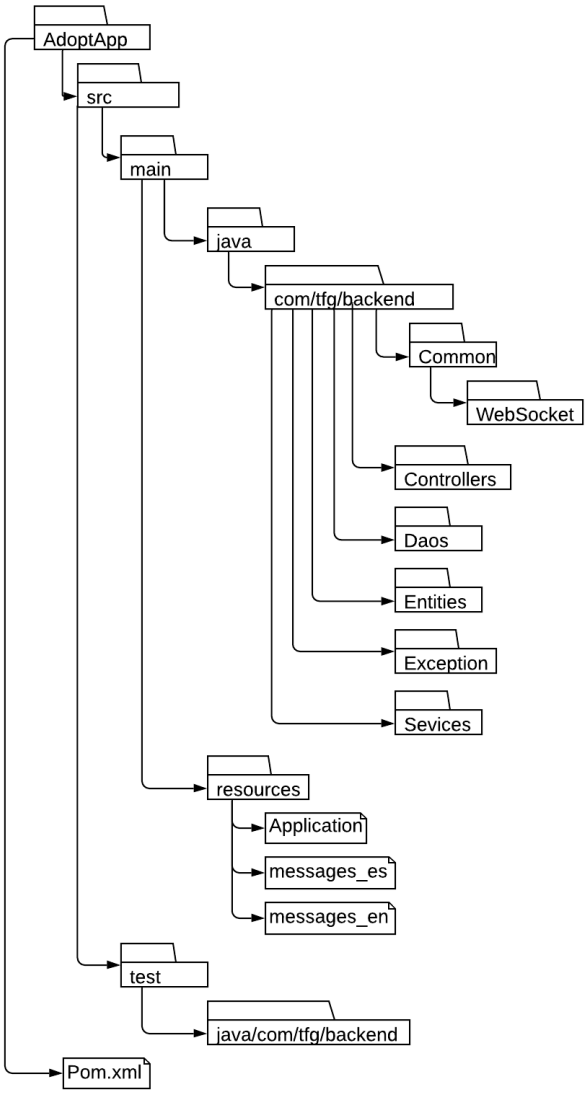
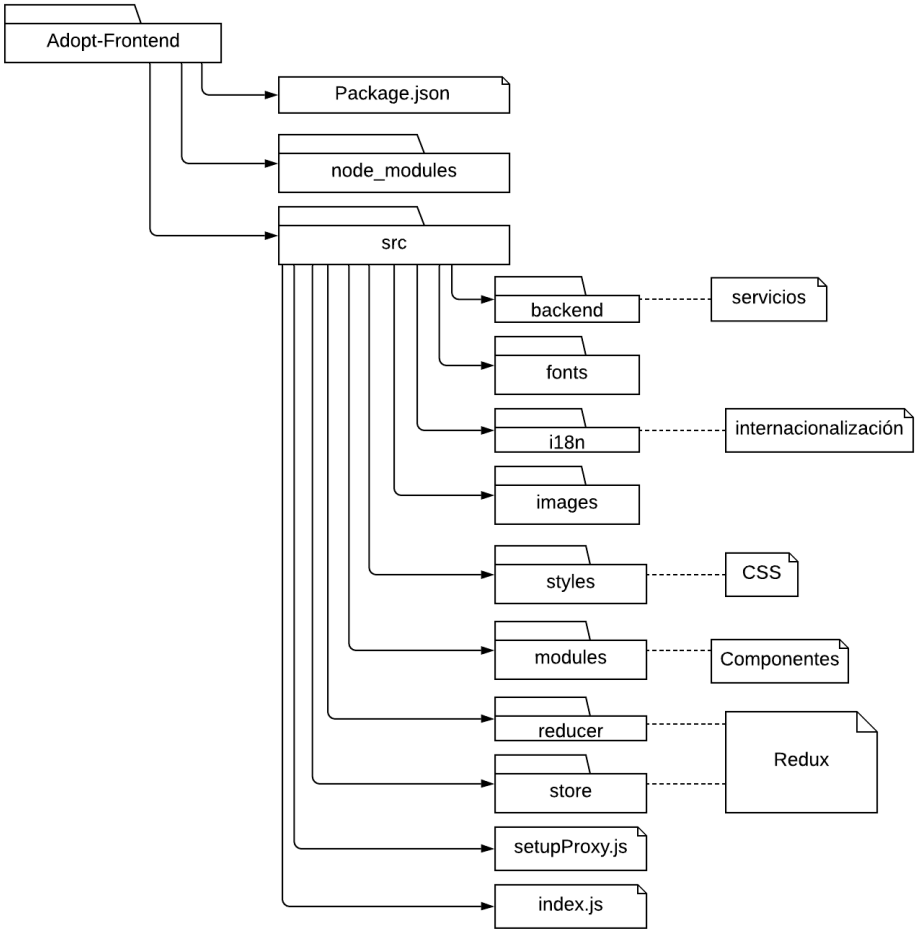


Figura 9.1: Estructura de ficheros del Backend

En lo que se refiere al *Backend*, por tratarse de un proyecto *Maven*[22], contiene un fichero *pom.xml* que incluye las dependencias y demás configuración del proyecto. El directorio *src*

contiene los ficheros que modelan las diferentes capas de la aplicación del lado servidor. Cabe destacar la existencia de ficheros de traducción en el directorio *resources*, así como la existencia del fichero *Application.properties* que contiene la configuración de *Springboot*[10].



S

Figura 9.2: Estructura de ficheros del Frontend

La estructura de ficheros del *Frontend* cuenta, en primer lugar, con un directorio *node modules* donde son descargadas las dependencias indicadas en el fichero *Package.json*. El directorio *src* contiene los ficheros que construyen la capa de servicios y la interfaz de usuario del *Frontend*. En el fichero *ServiceProxy.js* se configura la conexión al *Backend*.

9.3 Instrucciones de compilación

Para la compilación y ejecución del proyecto es necesario, en primer lugar, crear la base de datos necesaria. Los pasos a seguir son los siguientes:

- Se inicia *Mysql* ejecutando el siguiente comando en una terminal: **$mysqld**
- Se accede al gestor de base de datos con : **mysqladmin -u root –password** Se introduce la contraseña del usuario *root* en caso de que se haya introducido durante la instalación de *Mysql*
- Se crean la base de datos correspondiente y su usuario administrador mediante la siguiente secuencia de comandos:
  - **create AdoptApp**
  - **create user ”adoptappadmin@localhost” identified by ’adoptappadmin’**
  - **grant all privileges on adoptapp to adoptappadmin@localhost with grant option**

A continuación se procede a iniciar el *Backend* de la aplicación. Desde la carpeta raíz *AdoptApp* se ejecuta : **$springboot:run**.

Por último, para iniciar el *Frontend* se ejecuta desde el directorio raíz *AdoptApp-frontend* : **$npm start**.

Capítulo 10

Pruebas

10.1 Introducción

Durante el proceso de desarrollo de cualquier sistema software es importante verificar que el software que se está construyendo satisfaga los requisitos indicados por el usuario previamente. Por otra parte, resulta de vital importancia buscar e identificar errores y regresiones de forma temprana. Con este objetivo resulta importante realizar pruebas que localicen estos errores permitiendo a los desarrolladores solucionarlos o implementar mejoras. Existen diferentes tipos de prueba en función del momento de desarrollo en el que se realicen y la parte del sistema que se pruebe. En este apartado se describen los diferentes tipos de prueba realizados durante el desarrollo de la aplicación.

10.2 Pruebas de Integración

Durante la implementación de la capa de servicios del *Backend* es importante realizar este tipo de pruebas. Estas operan sobre los servicios del modelo comprobando que sus funciones se ejecutan correctamente y devuelven los resultados esperados. Estas pruebas son de especial importancia por dos motivos principales:

- Comprueban que la comunicación entre los servicios y las capas inferiores se produce correctamente. Además se verifica el correcto funcionamiento de los diferentes niveles.
- Facilitan la detección de errores de forma temprana además de verificar si un cambio en la implementación de algún módulo de la aplicación produce errores en otra parte del sistema.

  Para implementar estas pruebas en la aplicación se ha utilizado ***JUnit***[11]. Este conjunto de bibliotecas permite su ejecución de forma automatizada. Se han creado las clases de test que se muestran en la figura 10.1. Estos comprueban el correcto funcionamiento de

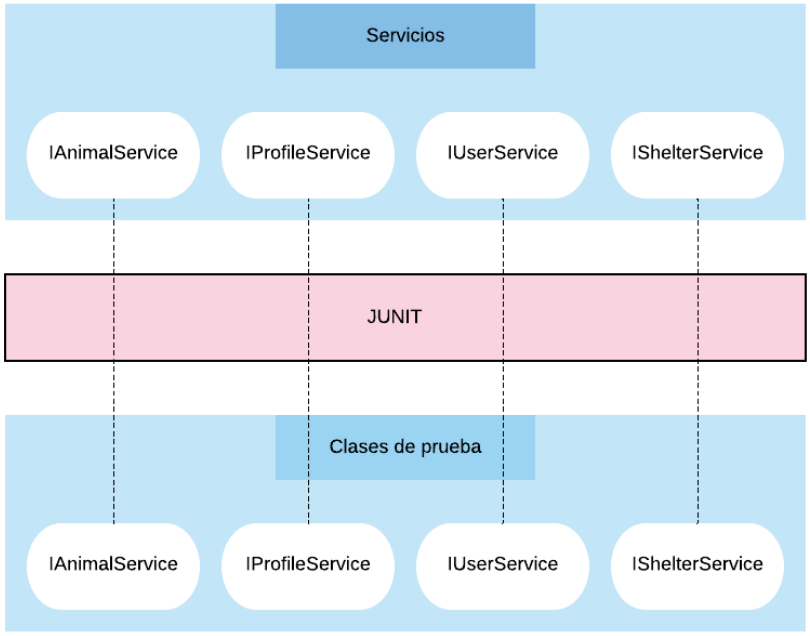


Figura 10.1: Clases de prueba y servicios probados

los servicios. Se anotan con *@Test* para ser identificados por *JUnit* y permiten verificar el resultado esperado ante la correcta ejecución de un servicio. También es necesario comprobar casos de fallo provocando ejecuciones que generan excepciones. En estos casos se añade la notación *ExpectedException* indicando la clase de excepción que se espera que produzca. Durante la ejecución de cada caso se crea un entorno de prueba, abriendo una nueva transacción en la BD. Para esto se anota la clase con *@Transactional*. Tras la ejecución y validación del resultado se hace *rollback* de la transacción, devolviendo a la BD a su estado previo.

10.3 Pruebas sobre la API REST

Con el objetivo de probar el correcto funcionamiento de la *Api Rest* se utiliza la herramienta *Postman*[29]. Esta proporciona una interfaz gráfica que permite simular peticiones *HTTP* que la aplicación cliente realiza sobre el servidor. De esta forma se comprueba que el resultado de estas peticiones es el esperado. Probando la *Api Rest* se prueban indirectamente las capas inferiores del *Backend* comprobando su correcto funcionamiento. En la figura 10.2 se puede ver el resultado de una prueba realizada con *Postman* en la que se solicita información de las distintas asociaciones registradas en la aplicación.

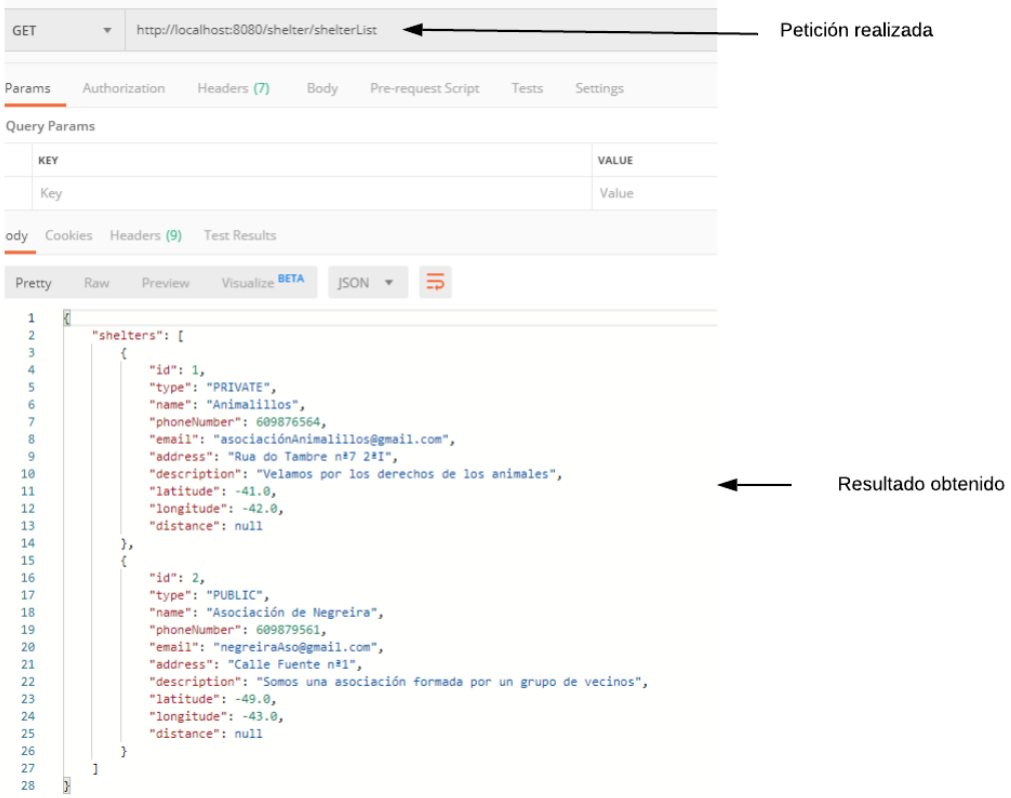


Figura 10.2: Prueba realizada en Postman

10.4 Pruebas de Aceptación

Tras completar el desarrollo del proyecto es necesario comprobar que la funcionalidad llevada a cabo cumple con los requisitos demandados. Para esto es necesario la ejecución de las funcionalidades a través de la interfaz de usuario. Estas pruebas son realizadas típicamente

por el cliente y suponen la ejecución de cada una de las funcionalidades del sistema. Algunos de ejemplos de pruebas de aceptación realizados en la aplicación son:

- Localización de animales perdidos :
  - Se registran dos usuarios individuales con localizaciones próximas entre sí.
  - Se accede al sistema con uno de ellos y se añade un animal perdido.
  - Se cierra sesión y se accede al sistema con el otro usuario creado.
  - Acceder a la lista de animales perdidos.
  - Se pulsa el botón *Localizar* y posteriormente el botón *Añadir* sin indicar una localización.
  - Se muestra un error indicando que se debe de introducir una localización.
  - Marcar una localización correcta y pulsar el botón *Añadir*.
  - Se cierra sesión y se vuelve a acceder al sistema con el usuario que añadió el animal.
  - Acceder a notificaciones, se muestra una notificación indicando que han localizado al animal.
  - Se accede a mis animales y se pulsa el botón *Localizaciones* del animal perdido.
  - La localización se muestra en el mapa y pulsando sobre ella se muestra la información correcta.
- Registro de asociación:
  - Acceder a la página de registro de una asociación.
  - Pulsar Añadir sin introducir ningún valor.
  - Se muestra un mensaje de error indicando que se deben introducir los campos obligatorios.
  - Introducir todos los campos obligatorios sin introducir una ”@” como parte del correo electrónico, se muestra un error indicando que el correo electrónico no tiene un formato válido.
  - Introducir los campos correctamente.
  - Se accede al sistema.
  - Cerrar sesión, registrar un usuario y acceder al sistema con el usuario registrado.
  - Acceder al listado de asociaciones, la información de la asociación se muestra correctamente.

La ejecución de las pruebas de aceptación constituye la ultima etapa del proyecto y una vez superada, este se considera finalizado.

Capítulo 11

Conclusiones y Futuras Líneas de Trabajo

11.1 Conclusiones

Para concluir se analizan los objetivos que se pretende alcanzar con la realización del proyecto y su nivel de cumplimento. Estos se pueden reducir fundamentalmente a construir un sistema que:

- Favorezca la reducción de animales en situación desamparo.
- Minimice la necesidad de tener que recurrir al mercado de animales de compañía.
- Aumente la visibilidad de animales en distintas asociaciones
- Facilite el proceso de localización de animales en adopción con ciertas características.
- Ayude en el proceso de localización de animales perdidos, fomentando la participación de los usuarios en el proceso.
- Facilite la comunicación entre usuarios

  Las funcionalidades planteadas e implementadas en el sistema desarrollado se enfocan a satisfacer los objetivos previamente nombrados. Puesto que todas las funcionalidades planteadas inicialmente han sido desarrolladas se puede concluir que los requisitos funcionales se han cumplido. Adicionalmente se pretendía desarrollar un sistema modular, con componentes independientes y que favoreciesen la incorporación de nuevos cambios, el mantenimiento y la comprensión del propio sistema. Estos objetivos se han tenido en cuenta en el diseño del sistema y se han aplicado durante su desarrollo.

11.2 Futuras líneas de trabajo

Partiendo de las funciones ya realizadas y teniendo en cuenta las carencias existentes en cuanto a plataformas que velen por el bienestar animal y faciliten diferentes procesos relacionados con estos, surgen nuevas funcionalidades que se pueden incorporar para satisfacer estas carencias. En particular se consideran las siguientes:

- Mejorar el componente social de la aplicación incorporando un foro que permita a los usuarios añadir temas y participar en ellos.
- Actualmente son las asociaciones las que añaden animales en adopción. En muchos casos puede darse la situación de que un usuario ya no pueda hacerse cargo de un animal. La aplicación se podría ampliar para tener en cuenta esta casuística y permitir a particulares dar animales en adopción.
- Incorporar un sistema de búsqueda que permita a las asociaciones localizar a aquellos usuarios que puedan estar interesados en alguno de sus animales en adopción.
- Permitir a las asociaciones añadir animales perdidos mediante volcado de datos desde distintos tipos de archivos, p.e *csv*. De esta forma se facilita el proceso de migrado de información desde otras plataformas a la aplicación desarrollada.

Apéndices

Apéndice A

Material adicional

A.1 Instalación del software

En este apartado se detalla el proceso de instalación y ejecución del software desarrollado. Para la ejecución de la aplicación es necesario tener instalado en primer lugar:

- Un sistema gestor de Base de Datos MySql 8.0 o superior
- Un servidor Apache

A continuación se crea la Base de Datos necesaria para la ejecución de la aplicación. Para ello se realizan los siguientes pasos:

- Ejecutar en un terminal : **mysqladmin-uroot–password** e introducir la contraseña del usuario root si se ha especificado durante la instalación de *Mysql*
- Ejecutar **create AdoptApp**
- Ejecutar **create user ”adoptappadmin@localhost” identified by ’adoptappadmin’**
- Ejecutar **grant all privileges on adoptapp to adoptappadmin@localhost with grantoption**

Para el despliegue del *Backend* :

- Abrimos una terminal en la ruta del archivo AdoptApp-0.0.1-SNAPSHOT
- Ejecutamos el comando : **$java -jar AdoptApp-0.0.1-SNAPSHOT**

Para desplegar el *Frontend* añadimos todos los archivos que lo conforman al directorio htdocs de la carpeta de *Apache*. Además incluimos en dicha carpeta un fichero *.htaccess* con el siguiente contenido:

Options -MultiViews
RewriteEngine On
RewriteCond %REQUEST_FILENAME !-f
RewriteRule ^index.html [QSA,L]

Posteriormente accedemos al fichero \conf\httpd del directorio de *Apache* y añadimos lo siguiente:

<Directory />
AllowOverride none
</Directory>

Por último publicamos los recursos del *Frontend* con el comando *httpd* desde la carpeta \bin del directorio de *Apache*

A.2 Manual de usuario

En este apartado se describen los pasos a llevar a cabo para realizar las distintas funcionalidades que proporciona el sistema. Para ello, estas se han dividido en cinco grupos:

- Acceso y Registro : Funcionalidades de registro y acceso a la aplicación.
- Perfil, Notificaciones y Preferencias : Gestión de perfiles, preferencias y notificaciones.
- Animales en Adopción : Funcionalidades de creación, modificación, borrado y consulta de animales en adopción.
- Animales Perdidos : Funcionalidades de creación, modificación, borrado y consulta de animales perdidos.
- Chat : Funciones relacionadas con la comunicación entre usuarios.
- Localización de animales perdidos : Añadir y consultar localizaciones de animales perdidos.

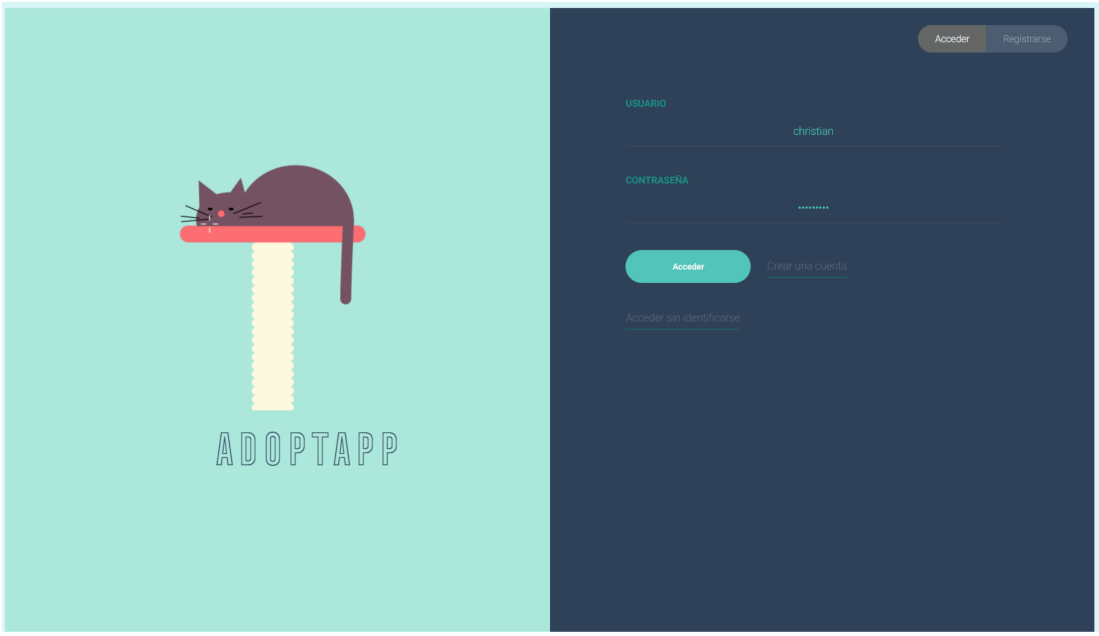


Figura A.1: Pantalla de Login

A.2.1 Acceso y Registro

En la figura A.1 se puede ver la pantalla de login, que se muestra al acceder a la aplicación. Aquí un usuario puede introducir sus credenciales para acceder al sistema como usuario autenticado. Si no se tiene cuenta de usuario se puede acceder pulsando *Acceder sin identificarse*. También se pueden acceder a la página de registro pulsando sobre *Registrarse*.

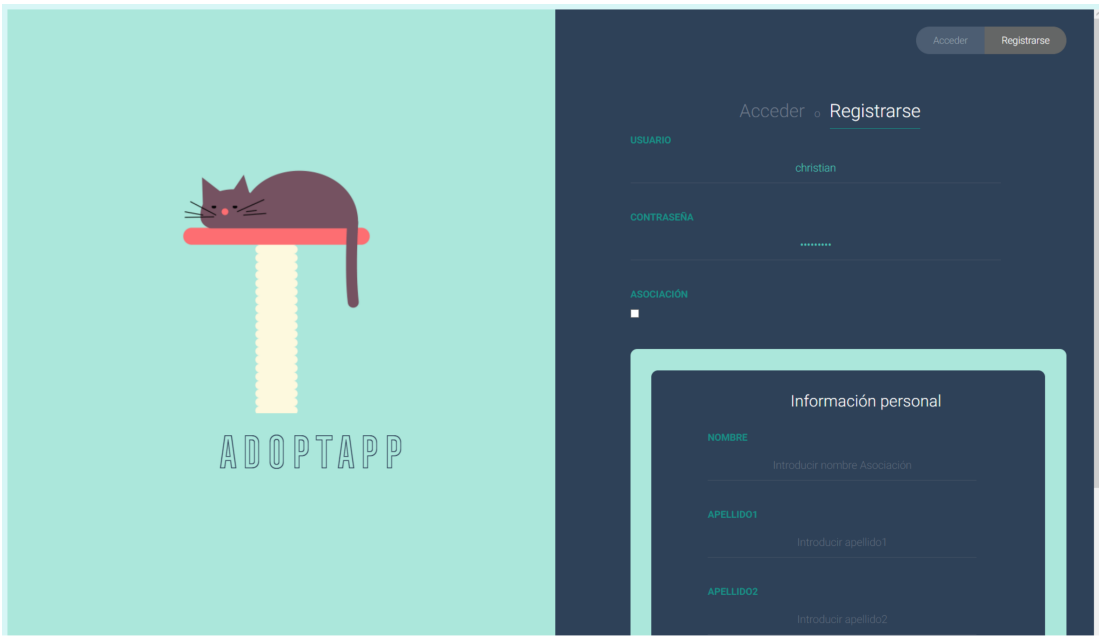


Figura A.2: Pantalla de Registro

En la figura A.2 se muestra la pantalla de registro de un nuevo usuario. A través de esta página se puede registrar un usuario individual o una asociación pulsando sobre la *checkbox* asociación.

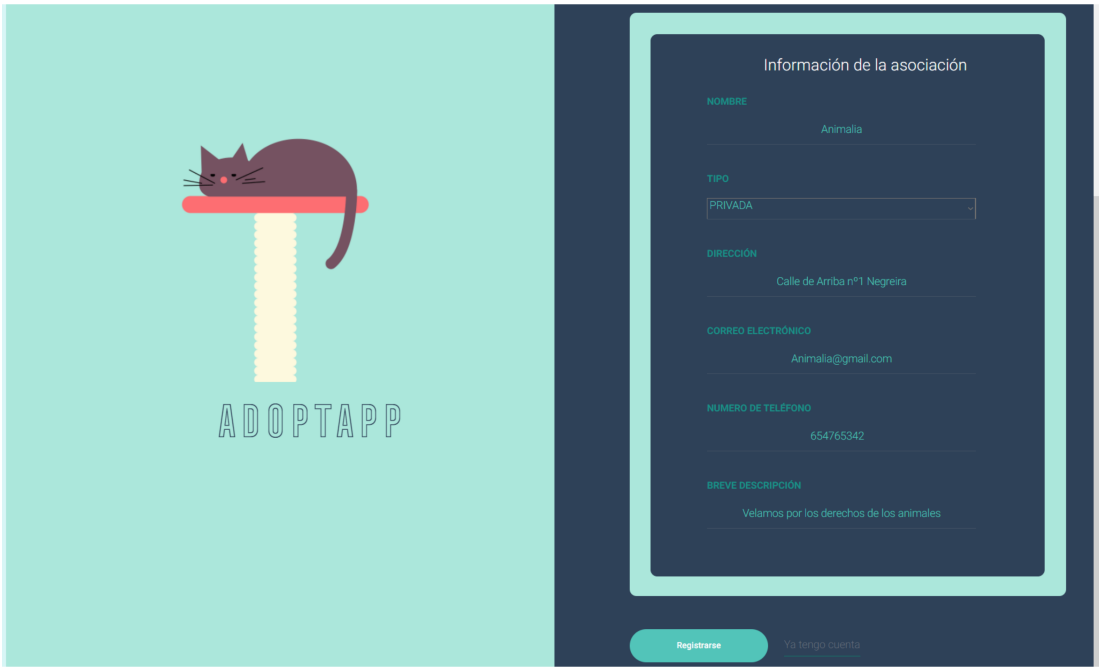


Figura A.3: Registro de asociación

Para registrar una asociación, es necesario rellenar la información que se muestra en la figura A.3 y pulsar sobre el botón *Registrarse*.

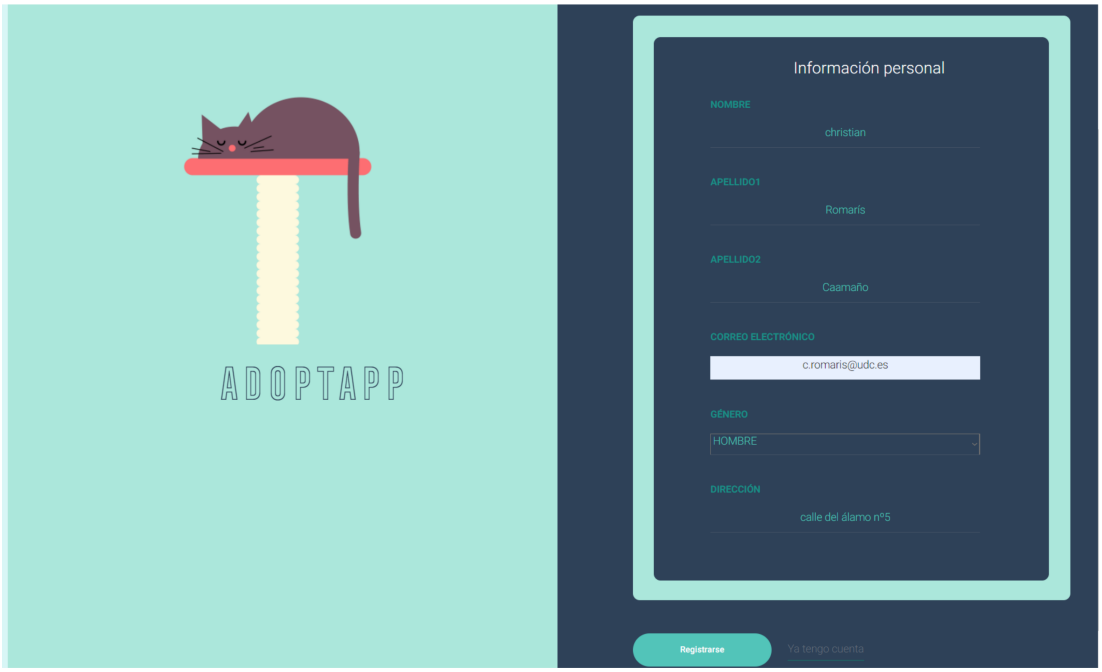


Figura A.4: Registro de usuario individual

Si, por otra parte, lo que se pretende es registrar un usuario individual será necesario rellenar la información mostrada en la figura A.4.

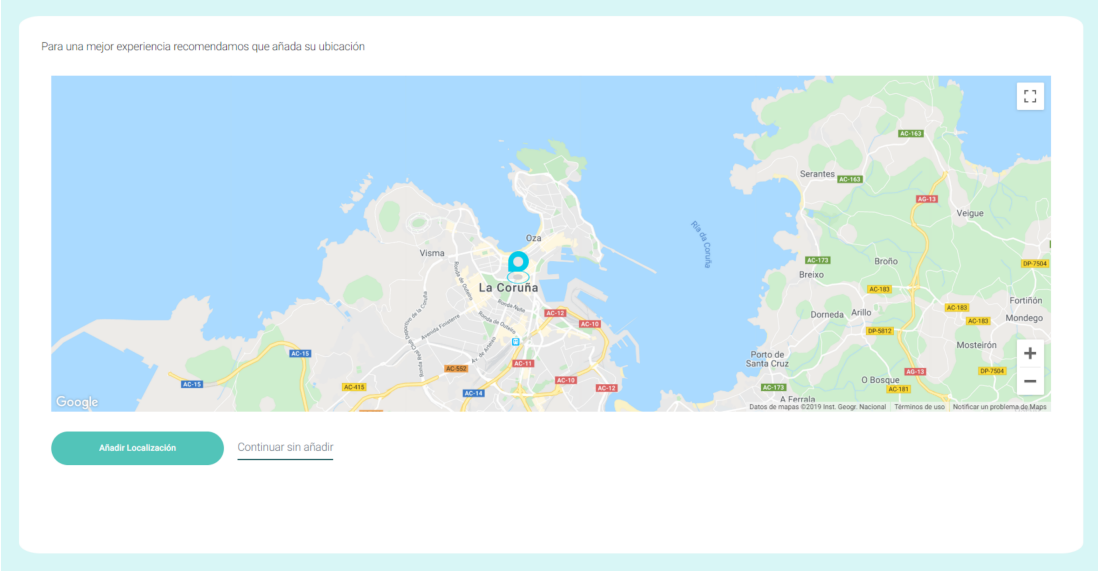


Figura A.5: Registrar Localización

Tras haber introducido la información de registro se muestra una página en la que el usuario puede indicar su localización como se muestra en la figura A.5 marcando un punto en el mapa y pulsando el botón *Añadir localización*. Si por el contrario no desea indicar una posición, puede pulsar sobre *Continuar sin añadir*.

Figura A.6: Menú de usuario anónimo

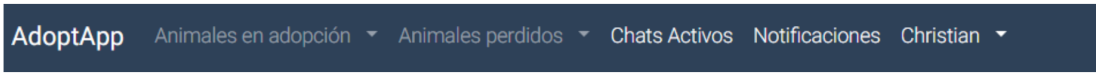


Figura A.7: Menú de Usuario individual

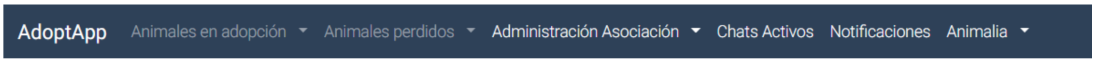


Figura A.8: Menú de asociación

Posteriormente se accede al sistema. Los usuarios tienen disponibles diferentes menús en función de si han accedido como usuario anónimo (Figura A.6), como usuario individual (Figura A.7) o como asociación (Figura A.8).

A.2.2 Perfil, Preferencias y Notificaciones

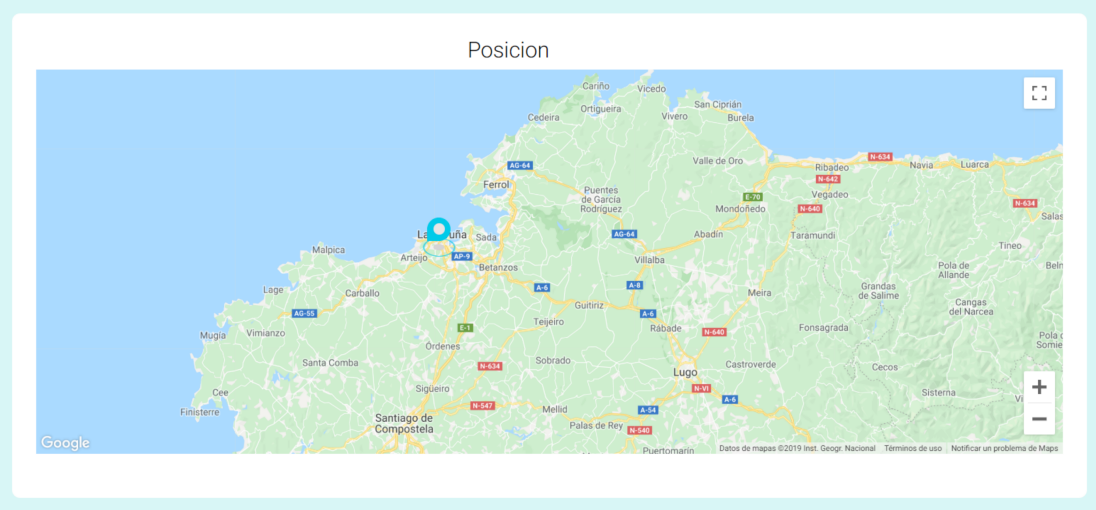


Figura A.9: Editar Localización

Para editar la información de perfil, es necesario pulsar sobre el nombre del usuario en el menú y posteriormente en *Información personal* o *Información asociación* dependiendo del tipo de perfil con el que se haya accedido al sistema. A continuación se muestra una página en la que el usuario puede modificar su información de perfil. Para modificar la localización, esta se selecciona en el mapa que se muestra en la figura A.9 y se pulsa el botón *Actualizar Información.*

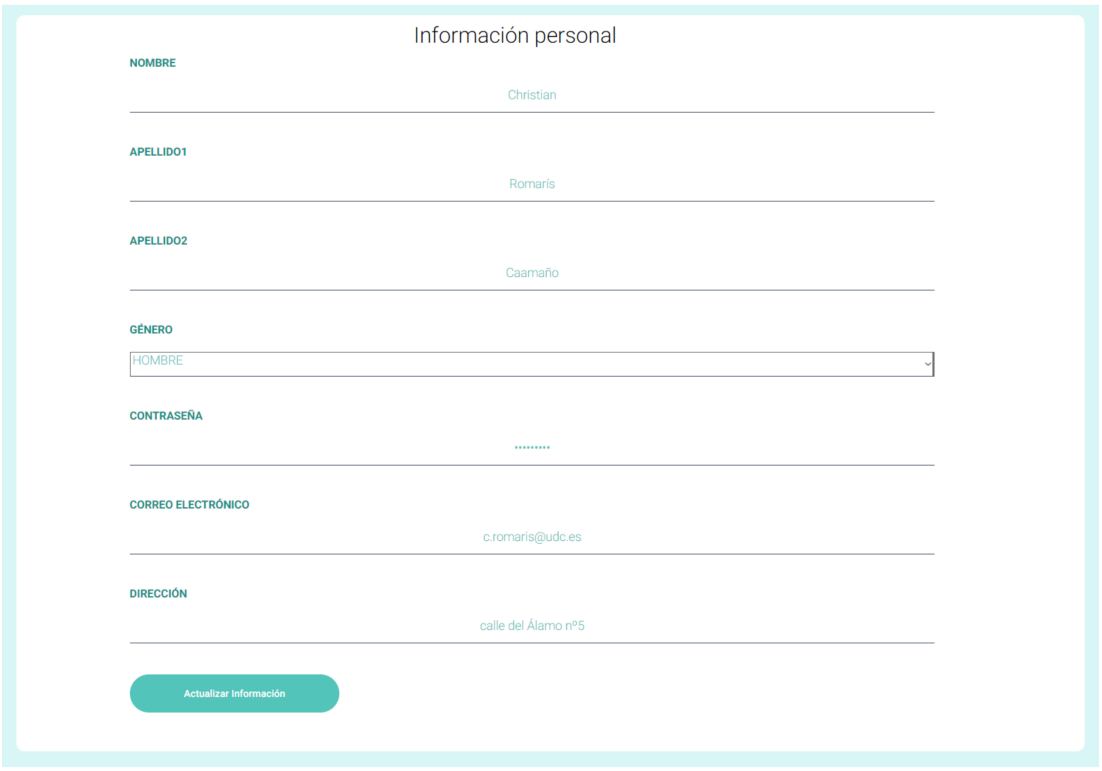


Figura A.10: Edición de información de usuario individual

Si un usuario individual desea modificar su información personal puede actualizar los campos que se muestran en la figuraA.10. Y pulsar sobre el botón *Actualizar Información.*

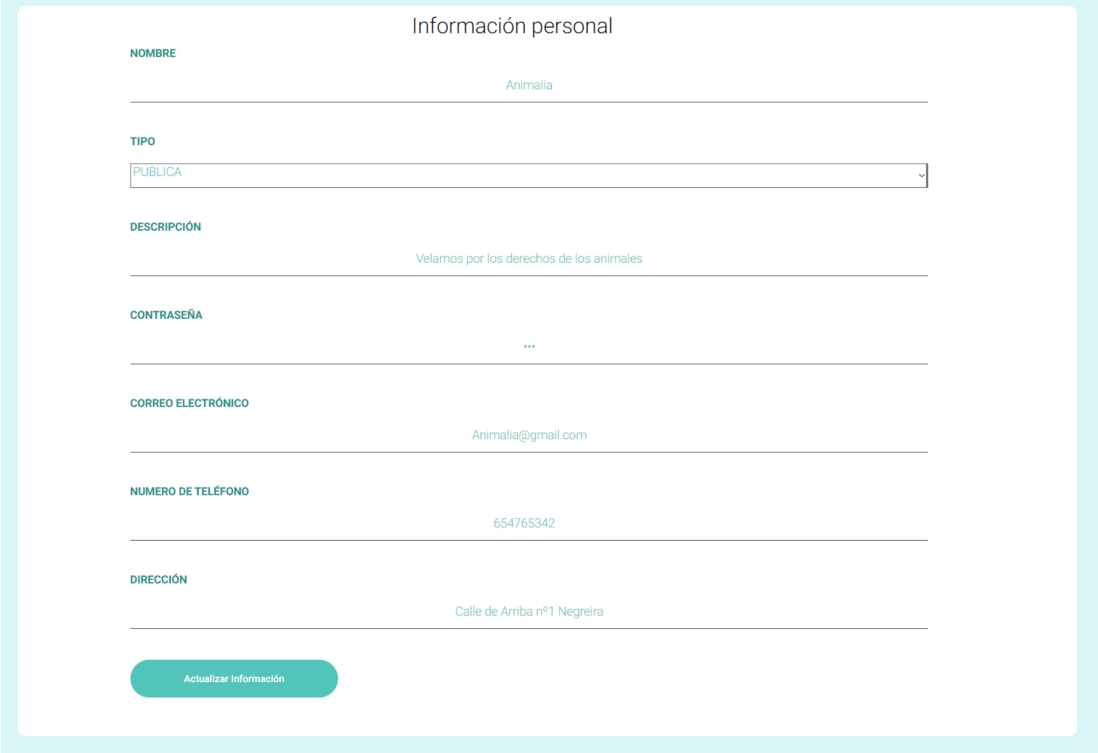


Figura A.11: Edición Información de asociación

En caso de que sea una asociación la que quiere modificar su información, puede modificar los campos mostrados en la figura A.11 y pulsar sobre el botón *Actualizar información*.

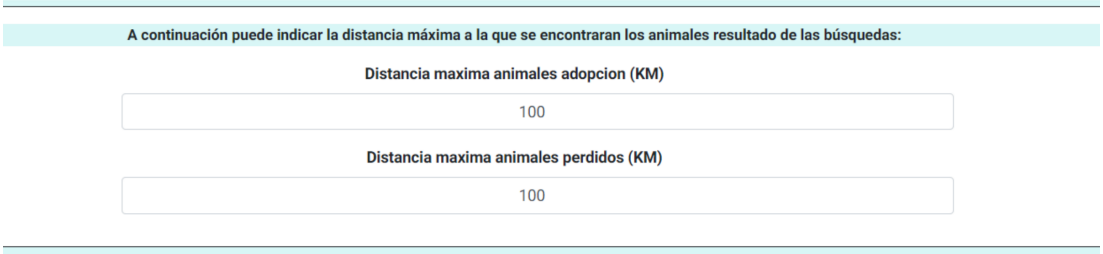


Figura A.12: Preferencias de distancia

Para modificar las preferencias del perfil seleccionamos el nombre del usuario autenticado en el menú y pulsamos *Preferencias*. Se muestran las preferencias del perfil correspondiente. Estas se pueden editar modificando los campos y pulsando sobre el botón *Actualizar*. Un usuario individual puede modificar las preferencias de distancia mostradas en la figura A.12, que indican la distancia máxima en la que se muestran asociaciones y animales perdidos en los mapas correspondientes. Puede modificar además las preferencias de animal mostradas en

la figura A.13 de forma que reciba notificaciones personalizadas en función de sus preferencias. También puede indicar si desea recibir un boletín semanal por correo electrónico como se puede ver en la figura A.14. Un perfil de tipo asociación únicamente puede modificar sus preferencias de distancia.

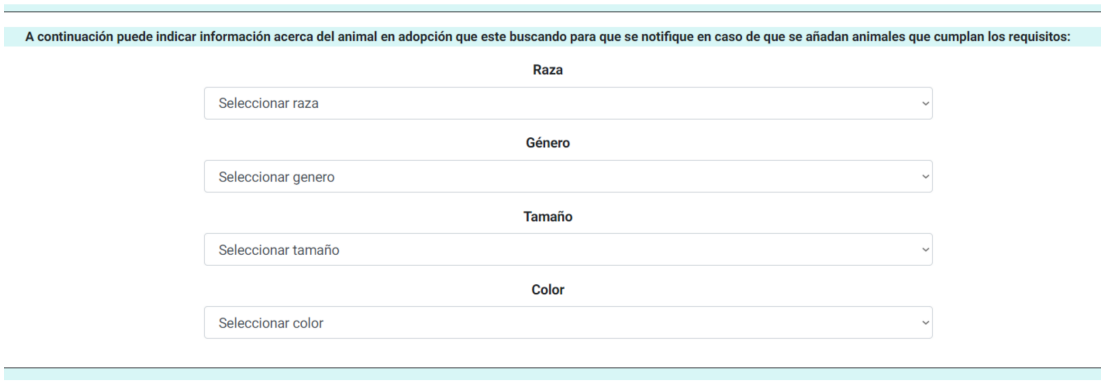


Figura A.13: Preferencias de animal

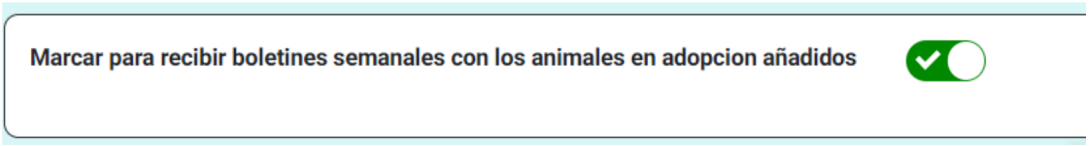


Figura A.14: Preferencias de boletín

Para consultar las notificaciones es necesario autenticarse en el sistema y pulsar sobre *Preferencias* en el menú. Se muestran las notificaciones del usuario como se muestra en la figura A.15. Pulsando sobre el botón *Ver Animal* se accede a la pagina de Información detallada del animal al que se refiere. Pulsando sobre el botón *Borrar* se elimina la notificación.

En caso de no tener notificaciones se muestra un mensaje como el de la figura A.16

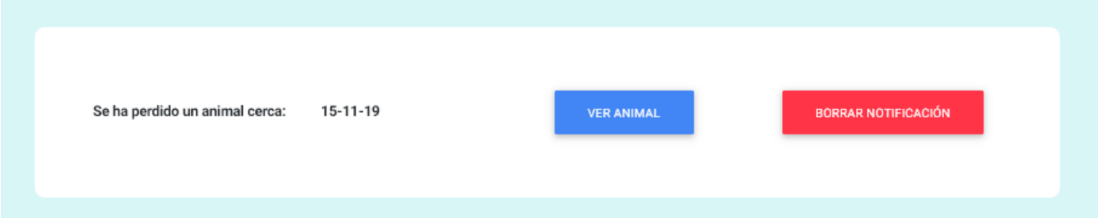


Figura A.15: Notificaciones

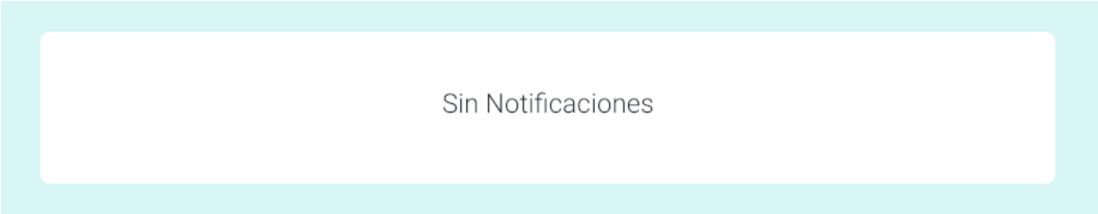


Figura A.16: Mensaje mostrado si no existen notificaciones

A.2.3 Animales en Adopción

Para añadir animales en adopción es necesario acceder a la aplicación con perfil de asociación. Una vez autenticado se selecciona en el menú *Administrar Asociación - Añadir animal* . Se muestra el formulario de creación de animales en adopción mostrado en la figuras A.17 y A.18. Se rellenan los campos y se pulsa el botón *Añadir*.

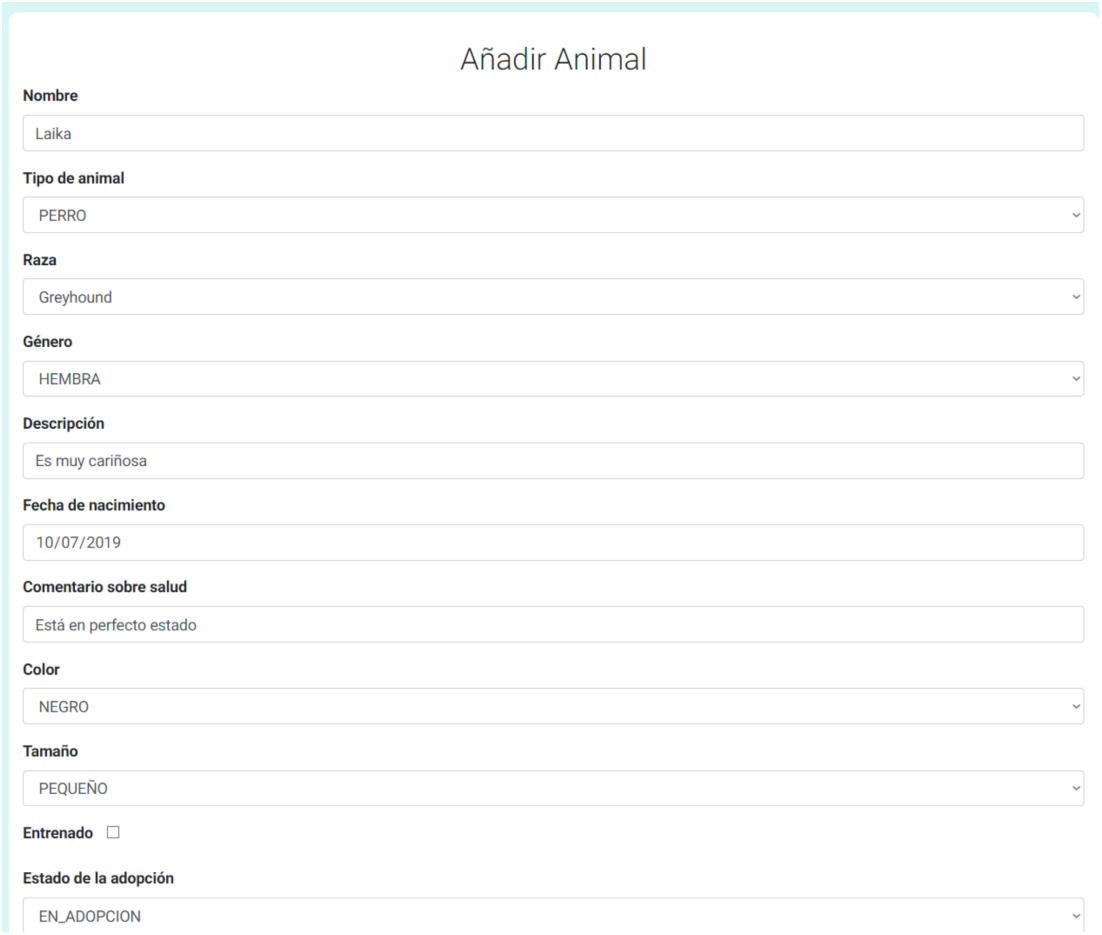


Figura A.17: Creación de animal en adopción 1

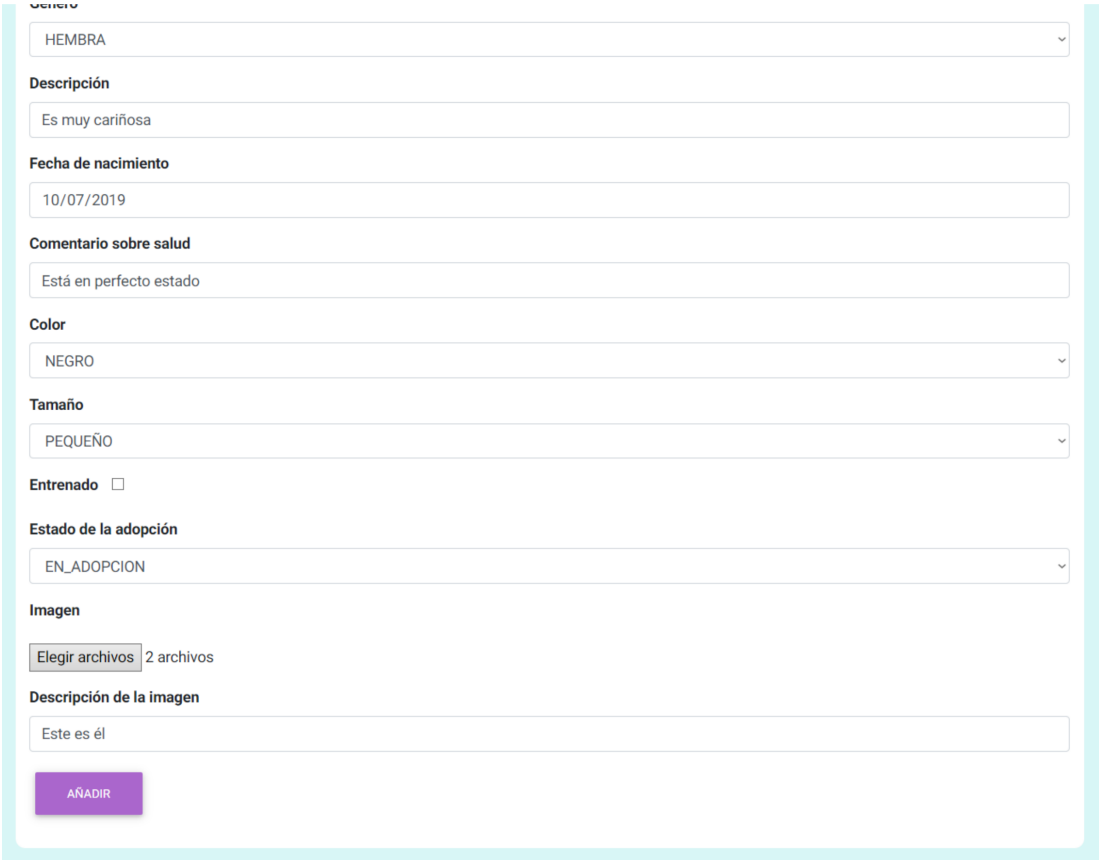


Figura A.18: Creación de animal en adopción 2

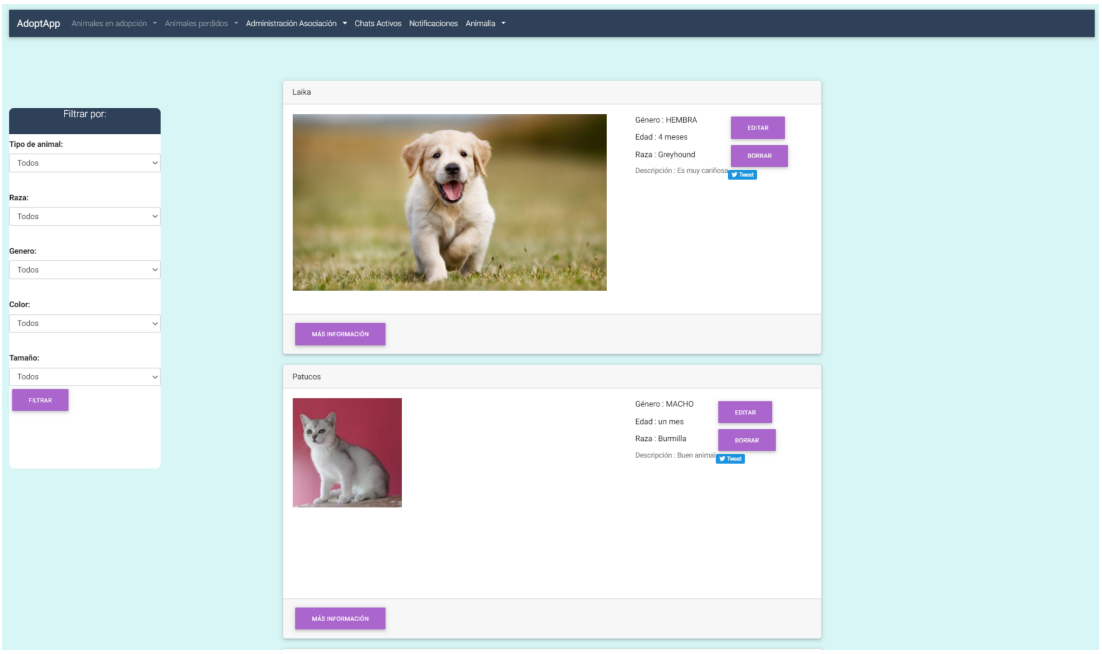


Figura A.19: Animales de la asociación

Para editar o borrar animales en adopción la asociación accede a *Administrar Asociación - Mis animales*. Se muestra el listado de animales en la asociación como se puede ver en la figura A.19, estos se pueden filtrar seleccionando las características del animal deseadas en el filtro lateral que podemos ver en la figura A.20 . Pulsando el botón *Borrar* se elimina el animal. Es posible compartir el animal en Twitter pulsando sobre el icono correspondiente. Para editar el animal se pulsa el botón *Editar* y se modifican los campos necesarios en el formulario de edición mostrado en A.21. Se pulsa el botón *Actualizar información* para persistir la información.

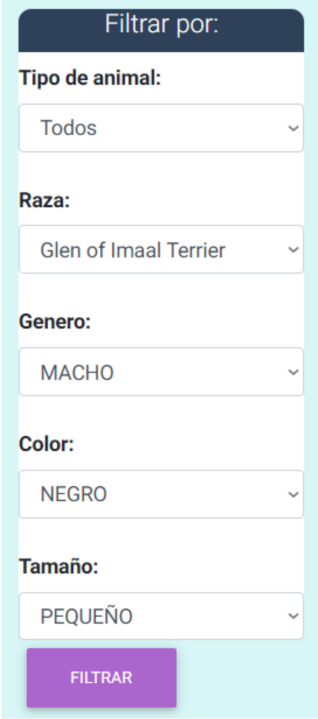


Figura A.20: Filtro

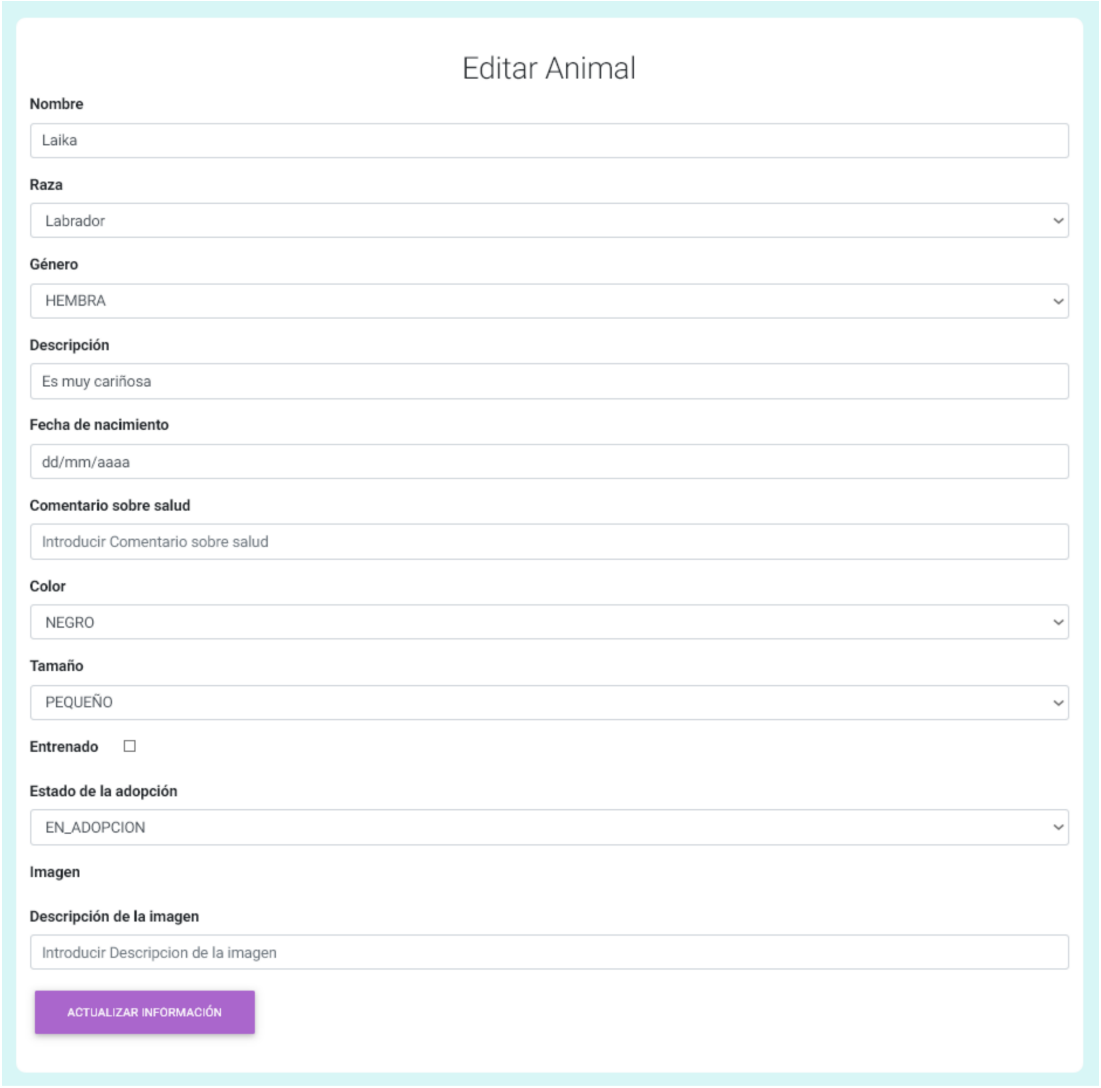


Figura A.21: Formulario de edición de un animal en adopción

Para consultar el listado de animales en adopción en las distintas asociaciones, cualquier usuario puede acceder a *Animales Adopción - Listado de animales* . Se muestra el listado completo de animales disponibles como se puede ver en la figura A.22, estos se puede filtrar.

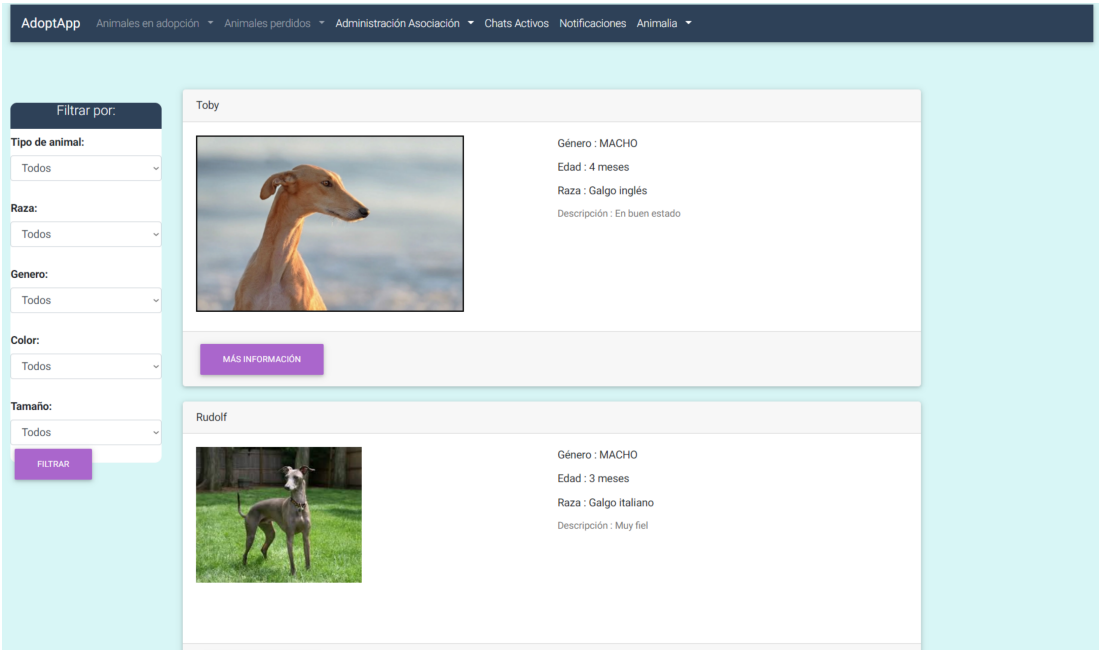


Figura A.22: Listado completo de animales en adopción

Pulsando sobre el botón *Mas información* de los animales se muestra información mas detallada del animal como se puede ver en la figura A.23.

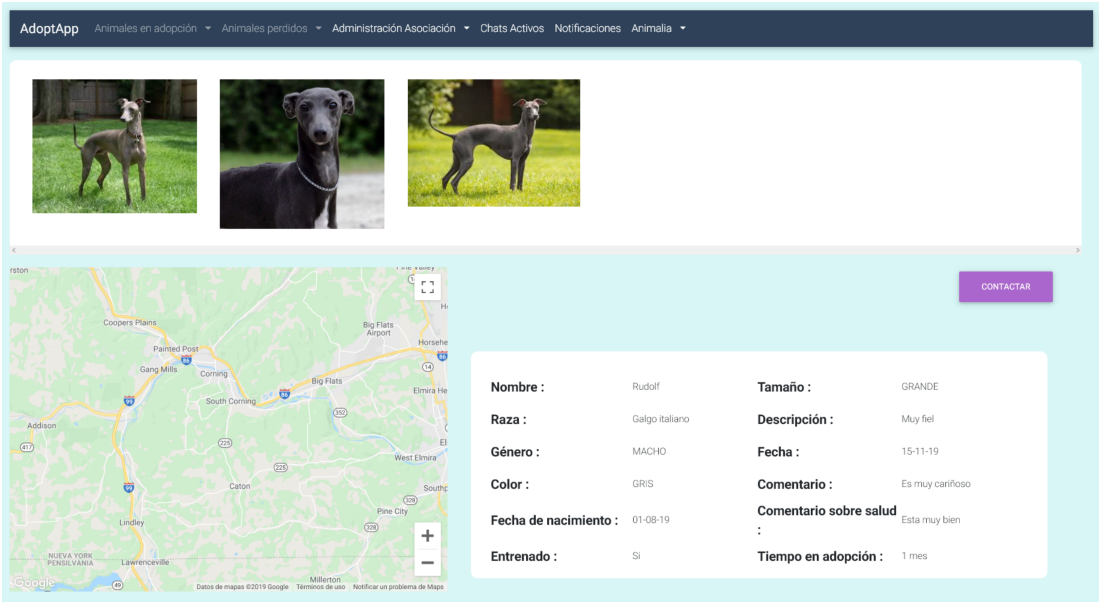


Figura A.23: Información detallada de un animal

Para visualizar la posición de las distintas asociaciones cualquier usuario puede acceder a *Animales Adopción - Mapa asociaciones*. Se muestra el mapa de la figura A.24 que indican la posición de las distintas asociaciones.

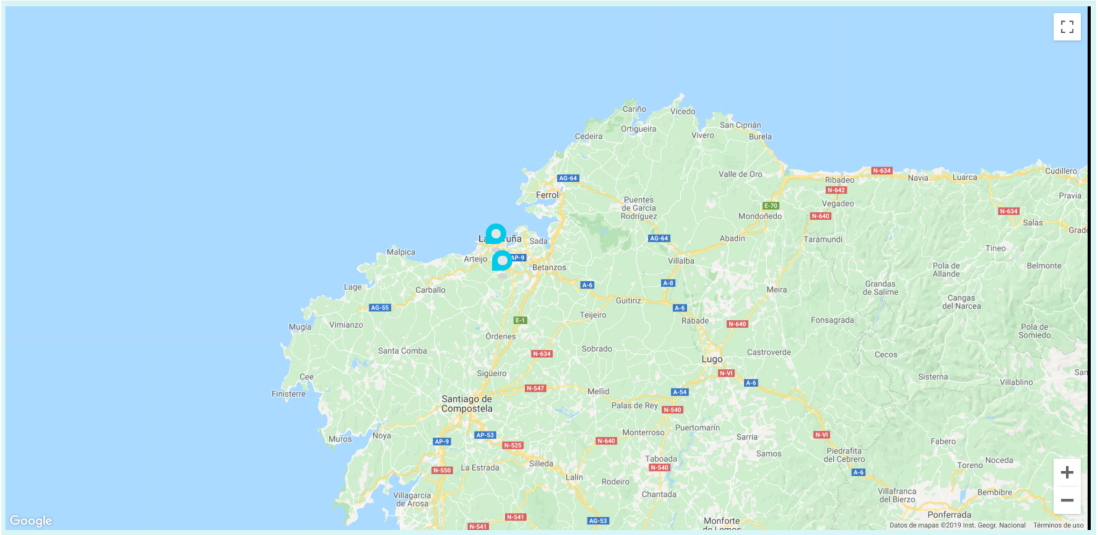


Figura A.24: Mapa de asociaciones

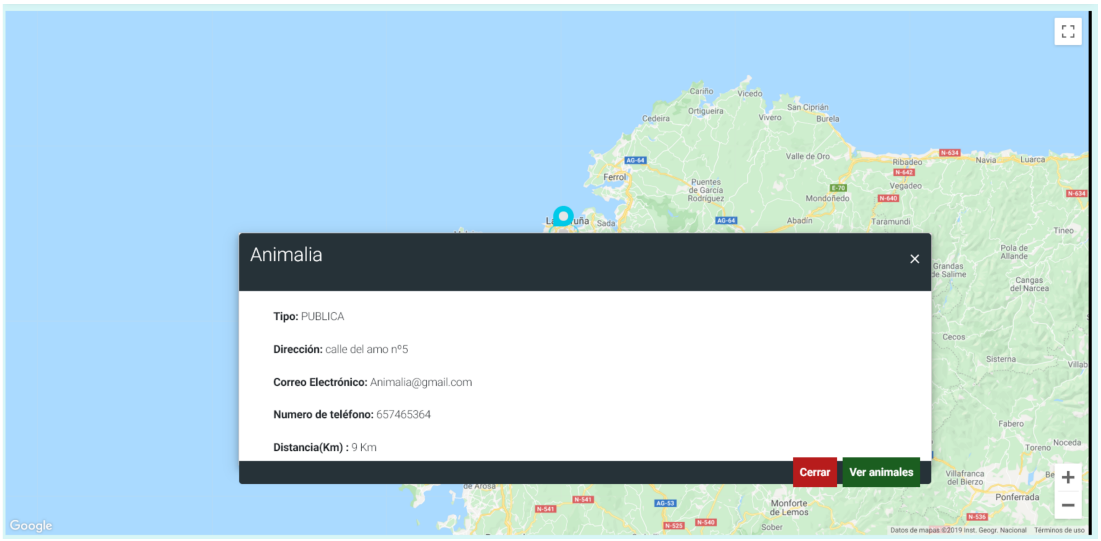


Figura A.25: Información de asociación

Pulsando sobre cada uno de estos marcadores se puede ver información de la asociación como se muestra en la figura A.25.
Para ver los animales en adopción disponibles en una asociación concreta se selecciona uno de los marcadores y se pulsa sobre *Ver animales*.

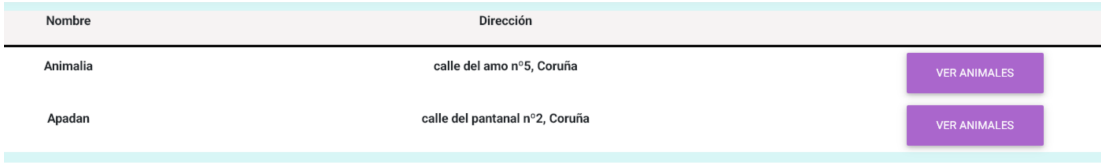


Figura A.26: Lista de asociaciones

Para consultar el listado completo de asociaciones cualquier usuario puede acceder a *Animales Adopción - Lista Asociaciones*. Se muestra el listado completo de asociaciones como se puede ver en la figura A.26. Pulsando sobre el botón ver animales de alguna de las asociaciones disponibles se muestra el listado de animales en la asociación.

A.2.4 Animales Perdidos

Para crear animales perdidos es necesario autenticarse con un perfil de usuario individual. Posteriormente se selecciona *Animales Perdidos - Añadir animal*. Se accede al formulario de creación de animales perdidos mostrado en las figuras A.27 y A.28 . Se indica la localización donde se ha perdido el animal, se rellena los campos solicitados y se pulsa el botón añadir.

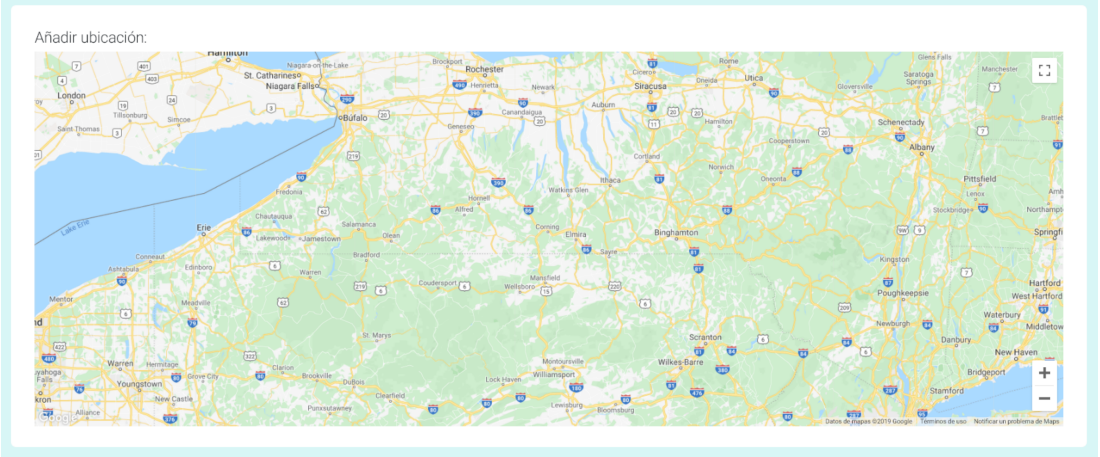


Figura A.27: Formulario de creación de animal perdido

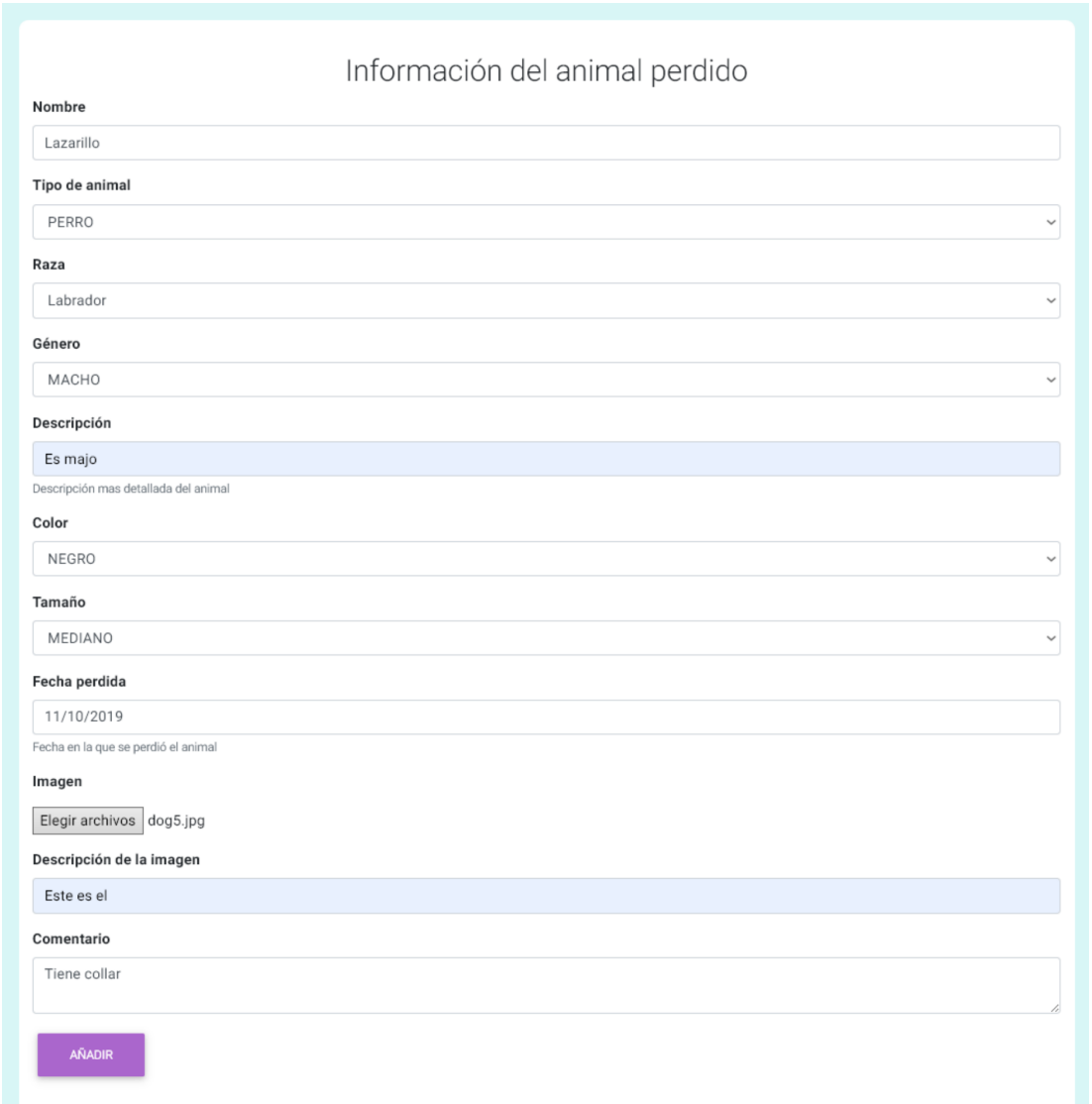


Figura A.28: Formulario de creación de animal perdido

Para editar o borrar la información de un animal perdido, el usuario accede a *Animales Perdidos- Mis Animales*. Se muestra el listado de animales perdidos del usuario como se ve en la figura A.29. Pulsando sobre el botón *Borrar* se elimina la información del animal. Para editar el animal se pulsa sobre el botón *Editar*. En este caso se muestra un formulario con la información del animal perdido como se puede ver en las figuras A.30 y A.31.

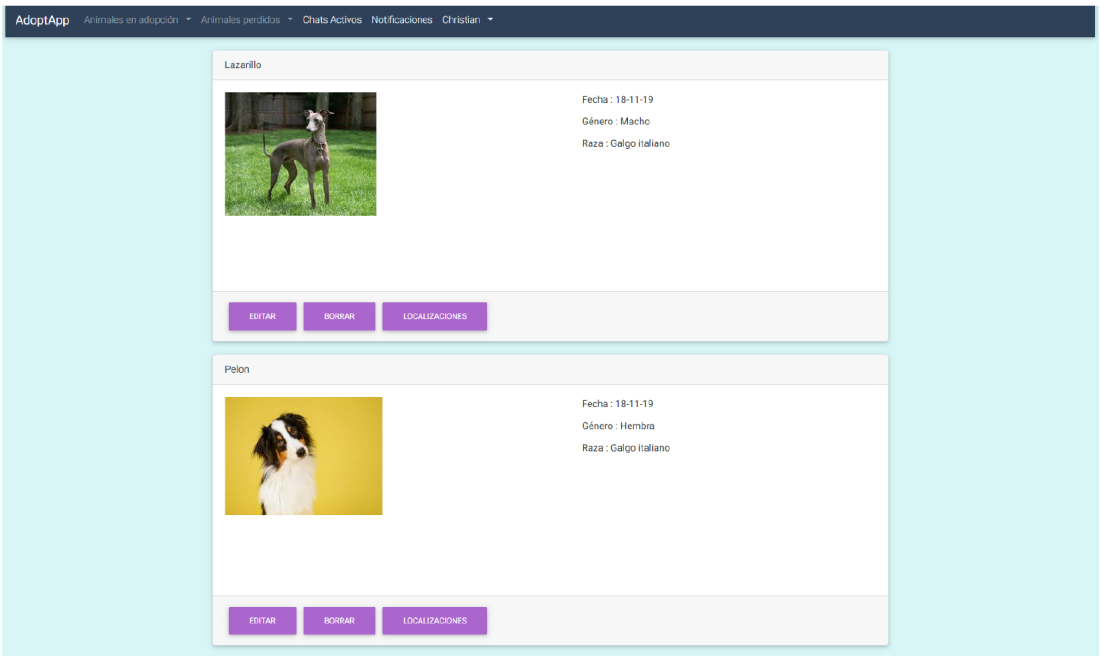


Figura A.29: Animales perdidos del usuario

La información es editable y pulsando sobre el botón *Actualizar información* se actualizan lo campos.

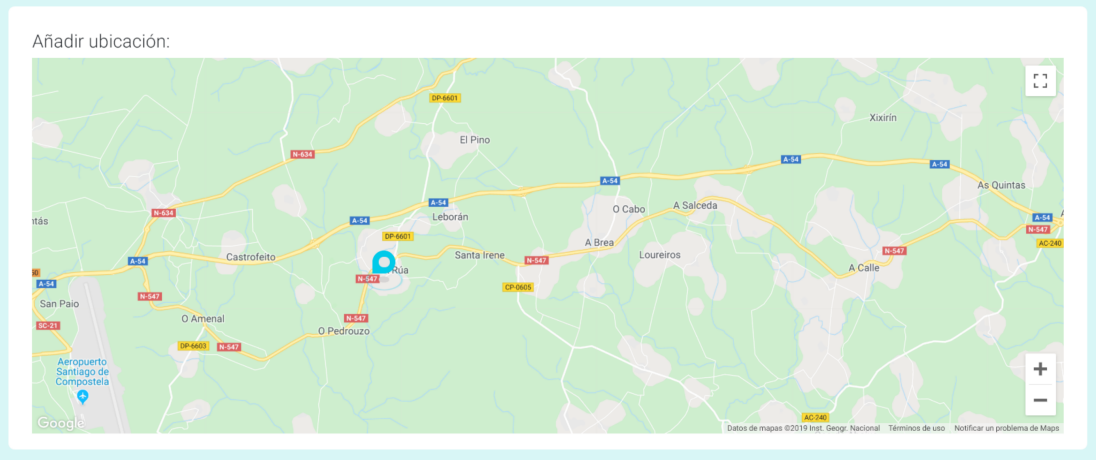


Figura A.30: Formulario de edición de animal perdido

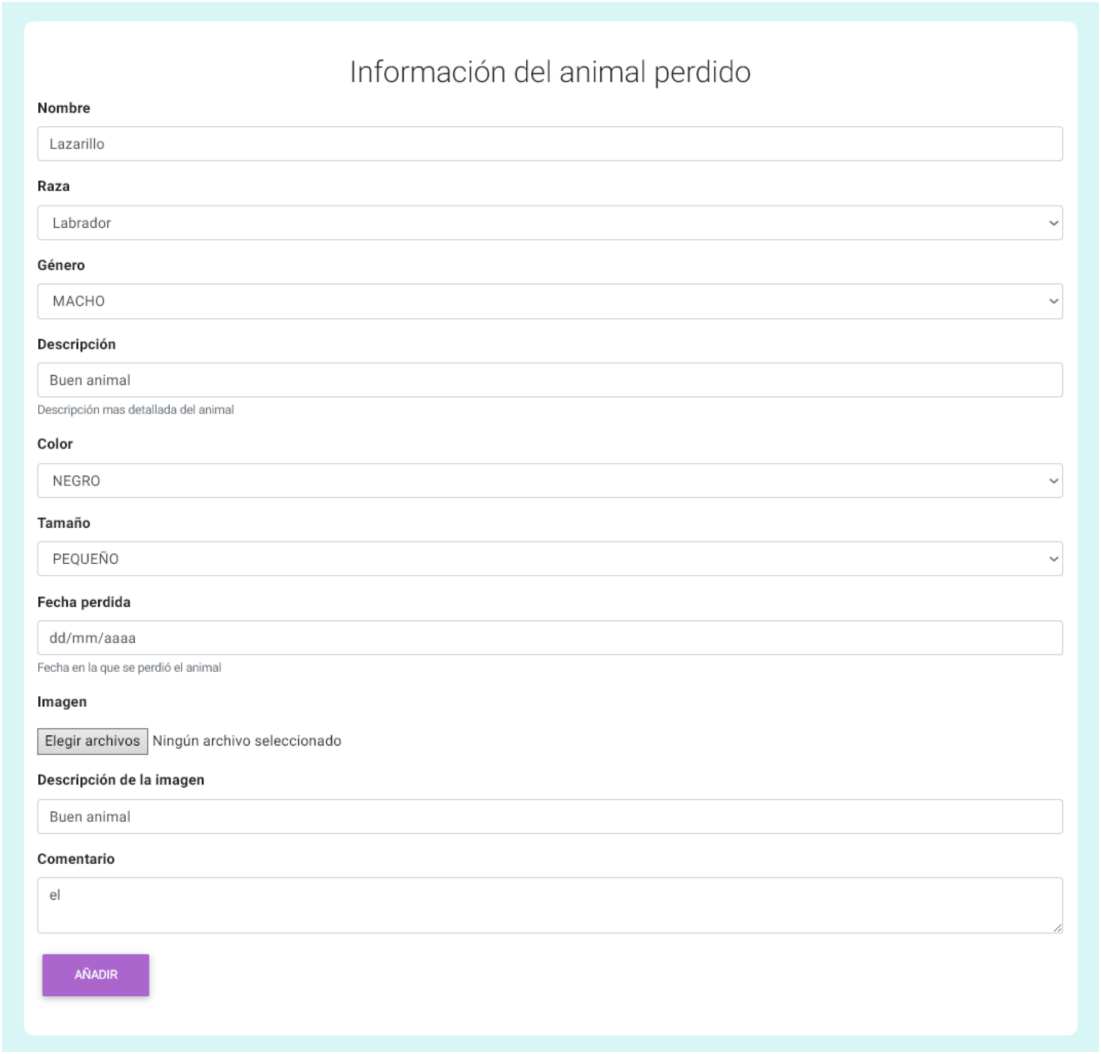


Figura A.31: Formulario de edición de animal perdido

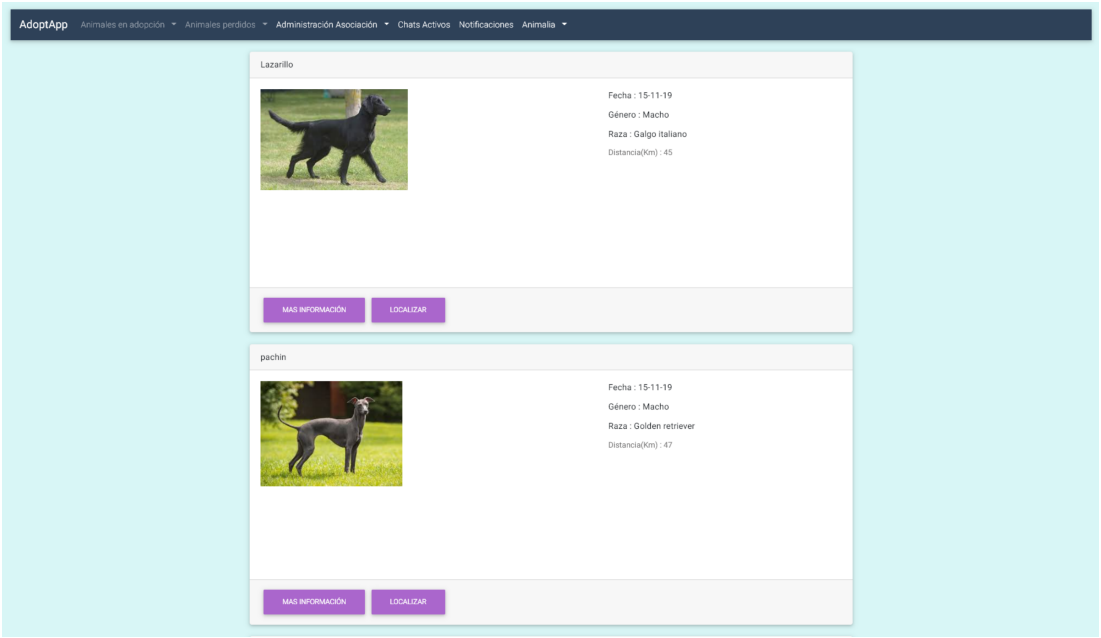


Figura A.32: Lista de animales perdidos

Cualquier usuario puede obtener el listado de animales perdidos. Para ello se accede a *Animales perdidos - Lista de Animales*, se muestra el listado de animales perdidos ordenados por distancia al usuario (en caso de que se haya indicado una localización de perfil) como se muestra en la figura A.32 . Pulsando sobre el botón *Mas información* se muestra información mas detallada del animal como se muestra en la figura A.33.

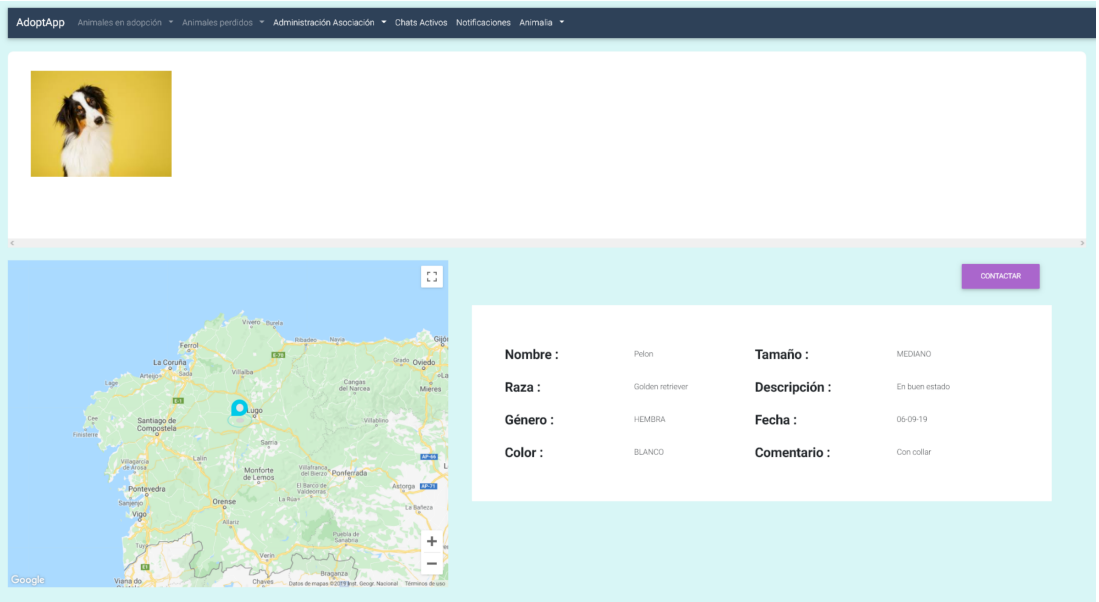


Figura A.33: Información de un animal Perdido

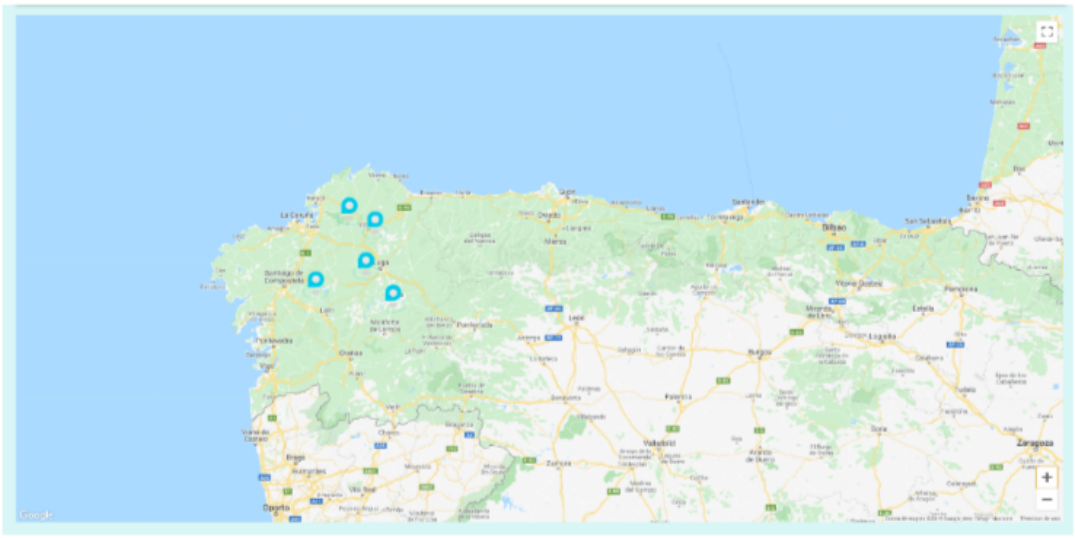


Figura A.34: Mapa de animales perdidos

Para consultar la posición de los distintos animales perdidos cualquier usuario puede acceder a *Animales perdidos - Mapa de animales*. Se muestra el mapa con distintos marcadores que simbolizan la posición de los distintos animales perdidos . Este mapa se puede ver en la figura A.34 Pulsando sobre cada uno de los marcadores se muestra información del animal como se muestra en la figuraA.35 y es posible consultar su información detallada pulsando sobre *Mas información*.

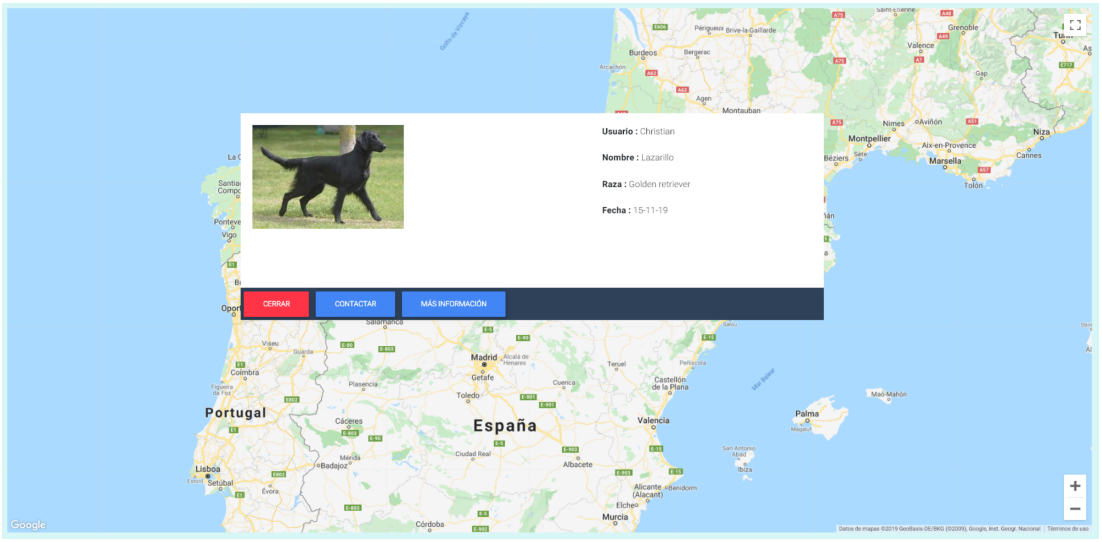


Figura A.35: Información de localización de animal perdido

A.2.5 Chat

Para contactar con un perfil, un usuario registrado accede a la página de información detallada de un animal en adopción mostrada en la figura A.23 o de un animal perdido mostrada en A.33 y pulsa sobre el botón *Contactar*. Se muestra un chat a través del cual pueden contactar ambos usuarios como se puede ver en la figura A.36. Para consultar los chats activos, un usuario autenticado puede seleccionar *Chats Activos* en el menú. Se muestra el listado de chats activos del usuario como se muestra en la figura A.37. Pulsando sobre el botón *Acceder* se muestra el chat con los mensajes previamente enviados entre ambos usuarios.

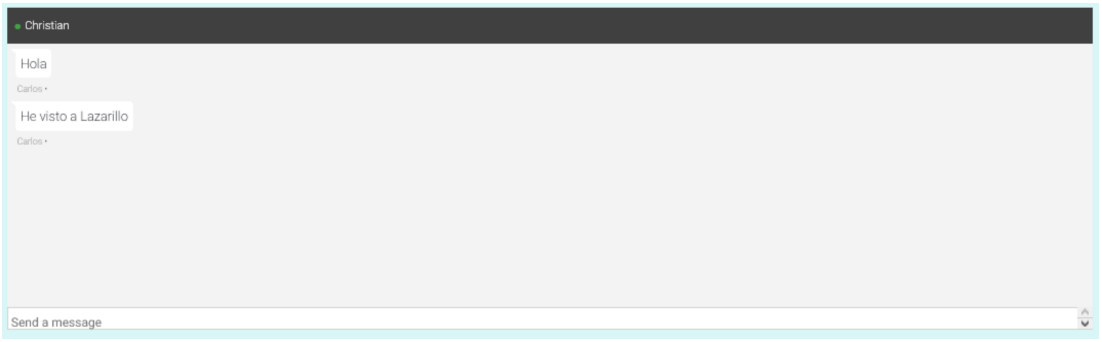


Figura A.36: Chat

Figura A.37: Chats Activos

A.2.6 Localización de Animales Perdidos

Un usuario anónimo o autenticado puede indicar la localización de un animal perdido. Para ello es necesario acceder al listado de animales perdidos que se muestra en la figura A.32 y pulsar el botón *Localizar* de alguno de los animales. Se muestra una ventana en la que el usuario puede indicar la localización en la que ha visto al animal como se puede ver en la figura A.38.

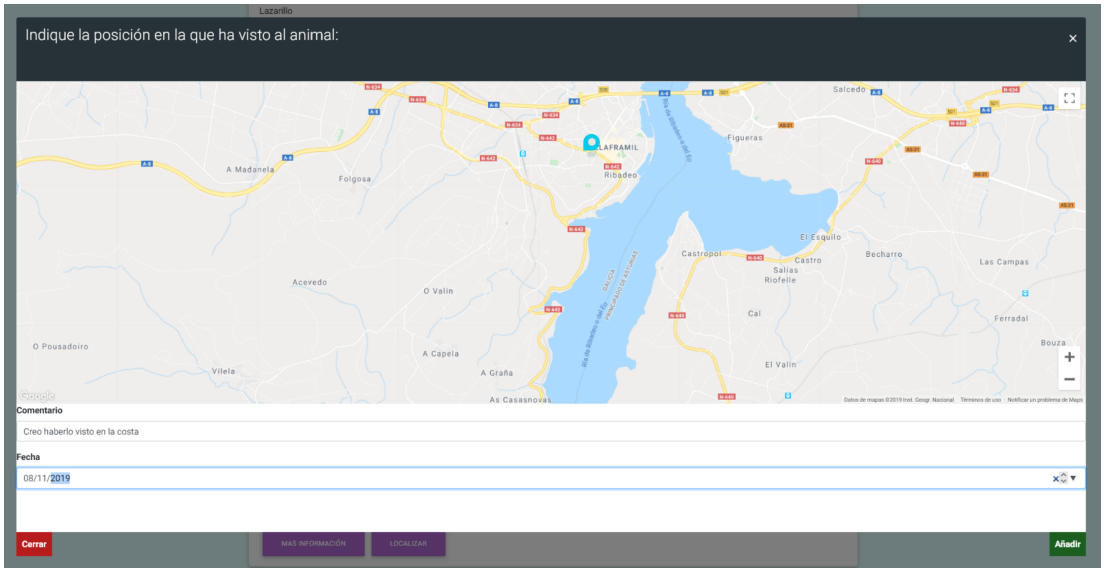


Figura A.38: Ventana para indicar la localización de un animal

El dueño de un animal perdido puede ver las localizaciones indicadas sobre cada uno de sus animales . Para ello accede a su listado de animales perdidos como se muestra en la figura A.32 y pulsa el botón *Localizaciones*. Se muestra un mapa con diferentes marcadores que simbolizan las posiciones en las que ha sido visto el animal. Este mapa se puede ver en la figura A.39. Pulsando sobre alguna de las localizaciones se muestra la información del usuario que marcó dicha localización como se muestra en la figura A.40 y es posible, si es un usuario registrado, contactar con él pulsando sobre el botón *Contactar*.

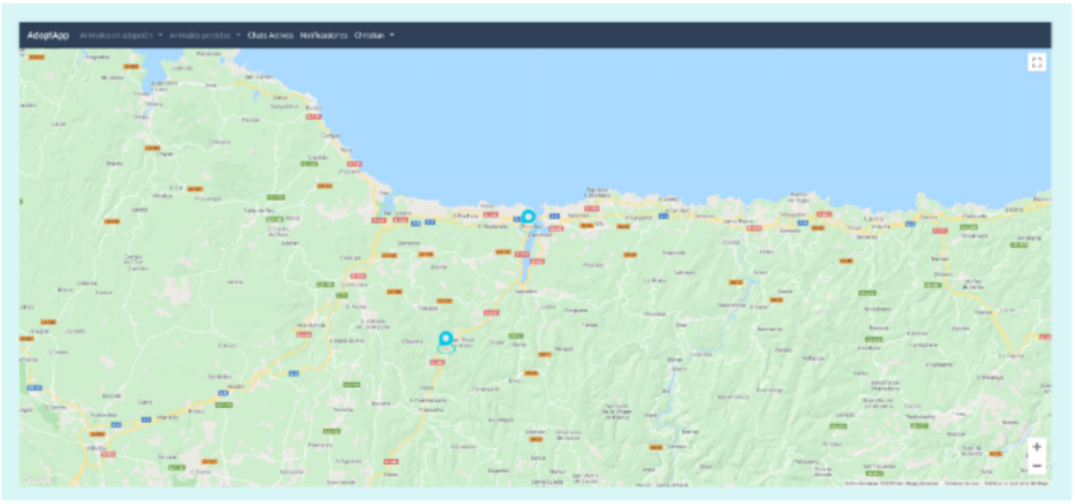


Figura A.39: Mapa con las localizaciones de un animal perdido

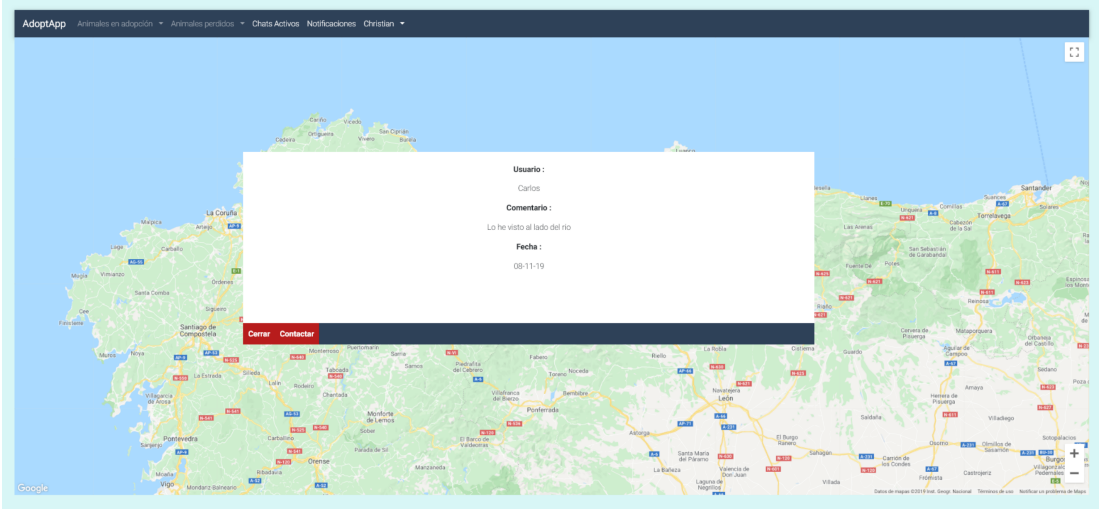


Figura A.40: Información de una localización

Lista de acrónimos

DTO *Data Transfer Object.*

DAO *Data Access Object.*

XML *Extensible Markup Language.*

JSON *JavaScript Object Notation.*

JDBC *Java Database Connectivity.*

HTML *HyperText Markup Language.*

CSS *Cascading Style Sheets.*

BD *Base de datos.*

HTTP *Hypertext Transfer Protocol.*

Glosario

Fronted Aplicación del lado cliente de una aplicación web.

Backend Aplicación del lado servidor de una aplicación web.

Framework Estructura conceptual y tecnológica de soporte definido, con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software.

Socket método de comunicación entre un programa cliente y un programa servidor en una red.

WebSocket API que permite realizar una comunicación bidireccional y abierta entre dos dispositivos.

Full-Duplex Termino utilizado para definir un sistema capaz de mantener una comunicación bidireccional

Bibliografía

[1] Apadan. Apadan. [Online]. Available: http://www.apadan.org

[2] A. Peludiños. Perfil de facebook de la asociación de animales de negreira. [Online]. Available: https://www.facebook.com/peludinassenfogar/

[3] MundoAnimalia. Sitio web de mundoanimalia. [Online]. Available: http://www.mundoanimalia.com/animales_en_adopcion

[4] Plataforma de animales perdidos. [Online]. Available: https://animales-perdidos.org

[5] Oracle. Introduction to java. [En línea]. Disponible en: https://docs.oracle.com/javase/tutorial/

[6] Jpa query structure (jpql / criteria). [En línea]. Disponible en: https://www.objectdb.com/java/jpa/query/jpql/structure

[7] MDN. What is javascript? [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Qué_es_JavaScript

[8] ——. Html. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTML

[9] ——. Css. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/CSS

[10] Spring. Springboot overview. [En línea]. Disponible en: https://spring.io/projects/spring-boot

[11] Junit. [En línea]. Disponible en: https://junit.org/junit5/

[12] React. Introduction to react. [En línea]. Disponible en: https://es.reactjs.org/docs/getting-started.html

[13] Getting started. [En línea]. Disponible en: https://getbootstrap.com/docs/4.3/getting-started/introduction/

[14] Introduction to material design. [En línea]. Disponible en: https://material.io/design/introduction/#principles

[15] reactstrap. [En línea]. Disponible en: https://reactstrap.github.io

[16] react-intl. [En línea]. Disponible en: https://github.com/formatjs/react-intl

[17] google-map-react. [En línea]. Disponible en: https://github.com/google-map-react/google-map-react

[18] moment. Introducction to moment. [En línea]. Disponible en: https://momentjs.com/docs/

[19] react-router-dom. [En línea]. Disponible en: https://www.npmjs.com/package/react-router-dom

[20] mdbreact. Getting started with mdbreact. [En línea]. Disponible en: https://mdbootstrap.com/docs/react/

[21] M. G. y Will Faurot. Redux in action.

[22] SonatypeCompany. Maven: The definitive guide.

[23] Ibm. [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SSQP76_8.7.0/com.ibm.odm.dserver.rules.res.managing/topics/con_javase_javaee_applis.html

[24] Travis docs. [En línea]. Disponible en: https://docs.travis-ci.com

[25] Github. [En línea]. Disponible en: https://github.com

[26] Eclipse. [En línea]. Disponible en: https://www.eclipse.org

[27] Npm docs. [En línea]. Disponible en: https://docs.npmjs.com/about-npm/

[28] Dbvisualizer. [En línea]. Disponible en: https://www.dbvis.com

[29] Postman. [En línea]. Disponible en: https://www.getpostman.com

[30] Git. [En línea]. Disponible en: https://git-scm.com

[31] Redmine. [En línea]. Disponible en: https://www.redmine.org

[32] Mysql. [En línea]. Disponible en: https://www.mysql.com

[33] H2 database. [En línea]. Disponible en: https://www.h2database.com/html/main.html

[34] spring. Springtools. [En línea]. Disponible en: https://spring.io/tools

[35] IBM. Jpa(java persistence api). [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/cwlp_jpa.html

[36] ——. Que es jdbc. [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/SSGU8G_11.70.0/com.ibm.jdbc_pg.doc/ids_jdbc_011.htm

[37] Mozilla. Http. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP

[38] ——. Websockets. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/API/WebSockets_API

[39] JSON. Json. [En línea]. Disponible en: http://www.json.org

[40] Puds. [En línea]. Disponible en: https://sites.google.com/site/businesscontrolesi/productos-software/metodologia-del-trabajo/pud

[41] Spring. [En línea]. Disponible en: https://spring.io